Web Services Security Using SOAP, WSDL and UDDI

by

Lin Yan

A Thesis

Submitted to the Faculty of Graduate Studies

in Partial Fulfillment of the Requirements

for the Degree of

MASTER OF SCIENCE

Department of Electrical and Computer Engineering

University of Manitoba

Winnipeg, Manitoba, Canada

THE UNIVERSITY OF MANITOBA

FACULTY OF GRADUATE STUDIES
*****
COPYRIGHT PERMISSION

Web Services Security Using SOAP,
WSDL and UDDI

BY

Lin Yan

A Thesis/Practicum submitted to the Faculty of Graduate Studies of The University of

Manitoba in partial fulfillment of the requirement of the degree

OF

MASTER OF SCIENCE

ABSTRACT

The Internet is beginning to change the way businesses operate. Companies are using the Web for selling products, to find suppliers or trading partners, and to link existing applications to other applications. With the rise of today's e-business and e-commerce systems, web services are rapidly becoming the enabling technology to meet the need of commerce. However, they are not without problems. While Web services are designed to pass through firewalls and have the capability to access Application Program Interfaces (APIs), they also come with security issues.

In this thesis, I present a viable approach for securing Web services. This approach adopts the Entrust/PKI (Public Key Infrastructure) and PKCS#7 cryptographic message syntax standard. This approach offers the capability to restrict access to an XML Web service to authorized users, and to protect the integrity and confidentiality of XML messages exchanged in a Web service environment. I make some comparison of this approach with existing standards and technologies for Web services security. Finally, a prototype based on the service-oriented architecture (SOA) is developed to validate the performance and effectiveness of the proposed approach.

ACKNOWLEDGEMENTS

I would like acknowledge foremost the constructive guidance of my advisor, Dr. Robert D. McLeod. Without his patience, knowledge and encouragement, this thesis could not have been completed.

Also, I would like to thank my father for his love, support, and encouragement in my life. Without his contribution for taking care of my son during the period of my graduate studies, it would be impossible for me to have completed my graduate study program.

Last but not least, I am grateful to Tao Wang, my husband, and Shi Yan, our son, whose support and understanding was also vital to the completion of my thesis.

TABLE OF CONTENTS

Chapter 1

Introduction

This thesis presents a viable approach for securing Web services. The purpose of this approach is to increase the security of transferring XML (Extensible Markup Language) messages over the Internet. This approach adopts the Entrust/PKI (Public Key Infrastructure) and PKCS #7 cryptographic message syntax standard. A comparison of this approach with existing standards and technologies for Web services security is also reported. As a prototype, a sample credit card Web service is developed with the help of UML (Unified Modeling Language) modeling, to demonstrate the performance and effectiveness of our approach.

In the remainder of this introductory chapter, I first introduce Internet concepts and Web services in particular in Section 1.1 and Section 1.2 respectively. Following that, I review the associated Web services standards including XML, SOAP (Simple Object Access Protocol), WSDL (Web Services Description Language), and UDDI (Universal Description, Discovery and Integration) in Section 1.3. Finally I discuss the motivation and objective of this thesis in Section 1.4.

1.1 The Internet

The Internet was conceived in the 1960s. Under the leadership of the U.S. Department of Defense's Advanced Research Project Agency (ARPA), it grew from a paper architecture into a small network (ARPANET) intended to promote the sharing of computers amongst researchers in the United States [Brief]. ARPANET was a success from the very beginning. The number of hosts connecting universities and government research centers was increased, and a commercial version of the ARPANET (Telenet) went online in the 1970s. In the 1980s, TCP/IP, the common protocol of all Internet computers was created by Bob Kahn and Vint Cerf. ARPANET has since evolved into the Internet as we know it today.

From the day it was created, the Internet has grown at an exponential rate [Paxson 1994]. According to Paxson, both the number of hosts connected to the Internet and the traffic in the Internet are growing at an exponential rate. Nowadays the Internet is beginning to change the way businesses operate. Companies are using the Web to sell products, to find suppliers or trading partners, and to link existing applications to other applications. The rapid proliferation of the Internet and the cost-effective growth of its key enabling technologies are revolutionizing information technology and creating unprecedented opportunities for developing large-scale distributed applications [Joshi et al. 2001].

1.2 Web Services

Web Services is an important new technology that allows applications on different platforms to exchange business data. Leading industry companies such as IBM, HP, Oracle, Microsoft, Novell, and Sun support Web Services by releasing their own Web Services products. There

are many Web Services platforms and Web Services development tools available today. Web Services will likely lead the development of new distributed business application solutions in the near future.

There are many definitions for Web Services. The World Wide Web Consortium (W3C)'s definition of Web Services states that "A Web Service is a software system identified by a URI whose public interfaces and bindings are defined and described using XML. As such, its definition can be discovered by other software systems. These systems may then interact with the Web Service in a manner prescribed by its definition, using XML-based messages conveyed by Internet protocols." A simpler definition is that a Web Service is a software application, accessible on the Web (or an enterprise's intranet) through a URL, that is accessed by clients using XML-based protocols, such as the Simple Object Access Protocol (SOAP) sent over accepted Internet protocols, such as HTTP. Clients access a Web Service application through its interfaces and bindings, which are defined using XML artifacts, such as a Web Services Definition Language (WSDL) file [Singh et al. 2004].

The main purpose of Web Service technologies is to allow applications on different platforms to exchange business data. Web Service technologies are used for Application-to-Application (A2A) integration or Business-to-Business (B2B) communication. A2A refers to disparate applications within a single organization communicating and exchanging data. B2B refers to multiple organizations, typically business partners, exchanging data. Web Service technologies today are used mostly in A2A, but they are also seeing growth in the B2B arena [Monson-Haefel 2004].

The word "Services" in Web Services refers to specific services which are part of a Service-Oriented Architecture (SOA). In a SOA, functionality is "published" on a network where two important capabilities are provided –"discovery", the ability to find the functionality, and "binding," the ability to connect to the functionality. In Web Services architectures, these activities correspond to three roles: Web Service provider, Web Service requester, and Web Service broker, which correspond to the "publish," "find," and "bind" aspects of a Service-Oriented Architecture. These components of a Web Services architecture are shown in Figure 1-1.

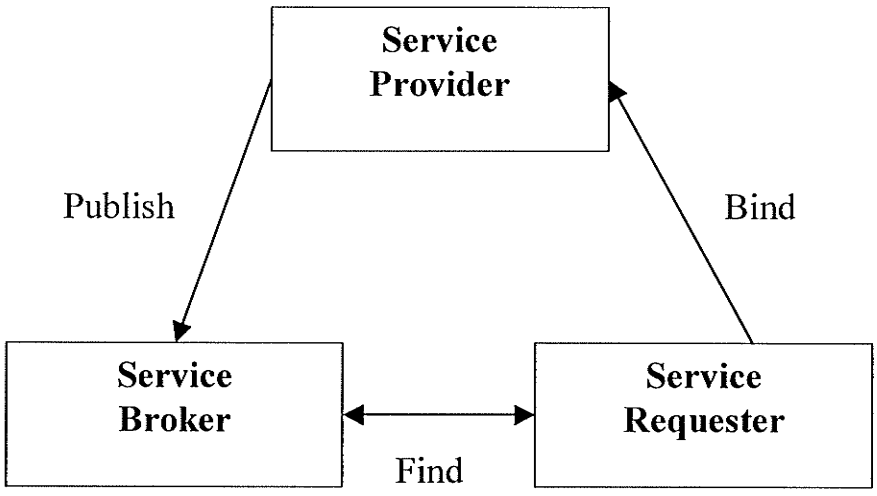

Figure 1-1 Web Services Architecture

1.3 Associated Web Services Standards

1.3.1 Extensible Markup Language - XML

XML was developed by an XML Working Group (originally known as the SGML Editorial Review Board) formed under the auspices of the World Wide Web Consortium (W3C) in 1996 [XML 2004]. XML is syntax to define markup languages. The advantages of XML are to allow structuring of documents in a standard way to make them easily machine-readable. XML is independent of operational platforms, service providers and requestors can communicate with each other in XML no matter what platform they use, and messages in XML can be communicated over the Internet using standard Internet protocols such as HTTP.

1.3.1.1 XML Example

XML is a simple, flexible, text-based markup language. XML data is marked using tags which we define the structure of the document. The tags reflect the meaning of the data they contain. Such markup allows different systems to easily exchange data with each other having only to agree on the tags to be used. This differs from tag usage in HTML, which is oriented to displaying data, while XML tags are oriented to describing data [Singh et al. 2004]. Listing 1-1 shows an XML document example.

```
<ContactInfo>
  <Name>Alice White</Name>
  <Address>
     <Street>20 Defoe Road</Street>
     <City>Winnipeg</City>
```

```
        <Province>Manitoba</Province>
        <Country>Canada</Country>
    </Address>
    <Phone>123-456-7890</Phone>
    <EMail>alice@cc.umanitoba.ca</EMail>
</ContactInfo>
```

Listing 1-1 Example XML Document

1.3.1.2 Document Type Defination - DTD

XML is based on SGML (Standardized General Markup Language). SGML includes a means of describing the structure of an XML document to define which particular elements and attributes (tags) are used and the order of those tags in an XML document. Such definition files are called DTDs – Document Type Definitions. DTDs allow systems which exchange XML documents to standardize their use of XML so that each system can understand the other's documents. Listing 1-2 shows a DTD that defines the XML used in the example in Listing 1-1.

```
<?xml version ="1.0"?>
<!DOCTYPE AddressingInfo [
<!ELEMENT ContactInfo (Name, Address, Phone, EMail)>
<!ELEMENT Name (#PCDATA)>
<!ELEMENT Address (Street, City, Province, Country)>
<!ELEMENT Street (#PCDATA)>
<!ELEMENT City (#PCDATA)>
<!ELEMENT Province (#PCDATA)>
<!ELEMENT Country (#PCDATA)>
<!ELEMENT Phone (#PCDATA)>
<!ELEMENT EMail (#PCDATA)>
```

]>

Listing 1-2 Example Document Type Definition (DTD)

This DTD opens with a line to state which XML version it supports. Then, the DOCTYPE directive states that this defines a piece of data called AddressInfo. The next line tells us that the ContactInfo contains four sub-elements: Name, Address, Phone, and Email. These are defined on the next three lines as being PCDATA. PCDATA means "parsed character data," meaning that they are to contain textual data that can be parsed by a text parser.

We can see from Listing 1-2 that DTDs define XML documents but do not actually use XML themselves. This is a weakness of DTDs. Furthermore, DTDs place no constraints on character data. Any data will be allowed in the XML document. DTDs don't support modularity and reuse.

1.3.1.3 XML Schema

XML Schema, an alternative schema language to DTDs, was invented to make up the weaknesses associated with DTDs. An example XML Schema is shown in Listing 1-3. As can be seen from the Listing, an XML Schema is expressed in XML and supports modularity. In addition, an XML Schema allows namespaces to be defined. A namespace is defined to enable using the same name with different meanings in different contexts. Since a namespace is specific to a particular context, each namespace is unrelated to any other namespace. Thus, the same name can be used in different namespaces without causing a duplicate name conflict [Singh et al. 2004].

```
<?xml version ="1.0" encoding="UTF-8"?>
<xs : schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

<xs : element name="ContactInfo">
    <xs : complexType>
      <xs : sequence>
          <xs : element name="Name" type="xs:string" use="required"/>
          <xs : element name="Address">
            <xs : complexType>
              <xs : sequence>
                  <xs : element name ="Street" type="xs:string"/>
                  <xs : element name ="City" type="xs:string"/>
                  <xs : element name ="Province" type="xs:string"/>
                  <xs : element name ="Country" type="xs:string"/>
              </xs : sequence>
            </xs : complexType>
          </xs : element>
          <xs : element name="Phone" type="xs: string"/>
          <xs : element name="EMail" type="xs:string"/>
      </xs:sequence>
    </xs:complexType>
</xs:element>

</xs:schema>
```

Listing 1-3 Example XML Schema

1.3.2 Simple Object Access Protocol - SOAP

SOAP is a lightweight protocol for the exchange of information in a decentralized, distributed environment. It is an XML based protocol that consists of three parts: an envelope that defines what is in a message and how to process it, a set of encoding rules for expressing instances of application-defined data types, and a convention for representing remote procedure calls and responses[Box et al. 2000]. SOAP can be used on top of any transport protocol, but HTTP is the popular protocol for transporting SOAP messages. SOAP will be discussed further in Chapter 4.

1.3.3 Web Services Description Language - WSDL

To be able to use a web service, we need to describe it, provide details for using it, and place it under the right category, similar to the Yellow Pages. The Web Services Description Language was developed to fulfill this requirement. It creates a standard way for specifying the details of a Web service. It is a general-purpose XML schema that can be used to specify details of Web service interfaces, bindings, and other deployment details [Singh et al. 2004]. With this standard way of specifying the details of a service, clients can discover and use that Web service even they have no prior knowledge of the service.

Ariba, IBM and Microsoft were involved in the development of WSDL 1.1 as a W3C Note for describing services for the W3C XML Activity on XML Protocols in March 2001. The first Working Draft of WSDL 1.2 was released by W3C in July 2002.

1.3.4 Universal Description, Discovery and Integration - UDDI

Universal Description, Discovery and Integration (UDDI) is a directory service where businesses and organizations can register, deregister and look up Web services. UDDI is a platform-independent framework for describing services, discovering businesses, and integrating business services by using the Internet [WSDL 2005]. Similar to the telephone system's yellow pages, a UDDI registry's sole purpose is to enable service providers to register their services and service requestors to find services. UDDI provides an interoperable, foundational infrastructure for a Web services-based software environment for both publicly

available services and services only exposed internally within an organization [Bellwood et al. 2002].

UDDI uses WSDL for the description of the web service interfaces and uses SOAP as the base protocol. The UDDI specification includes two categories of APIs for accessing UDDI services from applications: Inquiry APIs--enable lookup and browsing of registry information; Publishers APIs--allow applications to register services with the registry.

1.4 Motivation and Objectives

Making Web services accessible to clients over the Internet raises significant security issues, especially when these Web services are business critical. The Internet is already seen as a dangerous place, and the attributes of Web Services also introduce new potential vulnerabilities. By making access easier for legitimate users and systems from outside the enterprise boundary, new possibilities for malicious and felonious entities to exploit are also brought about [Kearney et al. 2004].

To prevent these new possibilities, a viable approach for securing Web services is presented in this thesis. This approach adopts the Entrust/PKI (Public Key Infrastructure) as the underlying security architecture and the PKCS #7 cryptographic message syntax standard to secure transferred data. As a prototype, a sample Credit Card Web Service is developed to validate the performance and effectiveness of the proposed approach. I also overview of the existing standards and technologies for Web services security. The main goals that the Credit Card Web Service prototype will achieve are:

Confidentiality	public and private key encryption will be used to protect the confidentiality of XML messages exchanged in the Credit Card Web Service environment.
Integrity	digital signatures will be used to prove that XML messages have not been tampered with in transit.
Non-repudiation	digital signatures will be used to bind the identity of a user to the transaction so that knowledge of the transaction cannot later be denied.
Accountability	digital signatures will be used to verify the identity of users

Summary

This chapter gave a brief introduction to Web services and corresponding standards and technologies, such as XML, SOAP, WSDL and UDDI. Security issue of Web services was presented, and the motivations and objectives of this thesis were outlined.

Chapter 2

Technologies for Web Services Security

In this chapter, we introduce some technologies related to Web Services security. I first provide an introduction to the PKI (Public Key Infrastructure). Following that, I review XML security including XML-Signature, XML-Encryption, and XKML (XML Key Management Specification). Finally I address WS-Security which is primarily used for securing SOAP messages.

2.1 Public Key Infrastructure - PKI

PKI stands for public key infrastructure. To get a better understanding of how a PKI provides security, I first introduce three underlying concepts: security through cryptography, certificates, and the Certification Authority.

2.1.1 Security through cryptography

We use a number of different types of keys to keep data secure. These include encryption key pairs, symmetric keys and signing key pairs.

2.1.1.1 Encryption Key Pair

An encryption key pair consists of a public key and a private key. The public key is used for encrypting data, known as the *public encryption key,* and the private key is used for decrypting

data, known as the *private decryption key*. Encrypting and decrypting data by using encryption key pairs is known as *public-key cryptography or asymmetric cryptography.*

A key that is used for both encrypting and decrypting data is called a *symmetric key*. This procedure is known as *symmetric cryptography*. We often use symmetric cryptography to secure large amounts of data due to its fast processing speed.

The following steps illustrate the process of using both symmetric-key and public key cryptography:

1. The sender encrypts the data with a one-time symmetric key, generated randomly for this step.
2. The sender encrypts the symmetric key with the recipient's public encryption key. Then, the sender forwards both the encrypted data and the encrypted symmetric key to its recipient.
3. After receiving the encrypted data and the encrypted symmetric key, the recipient first decrypts the symmetric key with his/her private decryption key. Note that, since the sender encrypted the symmetric key using the recipient's encryption public key, only the recipient's decryption private key can decrypt it.
4. The recipient uses the decrypted symmetric key to decrypt the data.

2.1.1.2 Signing Key Pair

A signing key pair provides a user with the means of generating a digital signature. A digital signature provides a guarantee to a recipient that signed data came from the person who signed

it, and that it was not altered since it was signed. A digital signature key pair is composed of a signing key (known as the *private signing key*) and a verification key (known as the *public verification key*).

To generate a digital signature, a sender follows these steps:

1. The sender first takes a mathematical summary (i.e. a *hash code*) of the data. This hash code is a uniquely identifying digital fingerprint of the data. If even a single bit of the data changes, the hash code will change.
2. Next, the sender encrypts the hash code with their private signing key. The sender then forwards the data and the encrypted hash code (that is, the signature) to the recipient.

Note that the encrypted hash code is an item that only the sender, using their private signing key, could have produced. A hash code is also called a *digest.*

The following steps describe the verification of the signature and confirmation that the data has not been altered since it was signed.

1. The recipient decrypts the hash code using the sender's public verification key to verify that the hash code was encrypted by the sender.
2. The recipient then uses the data to generate a new hash value using the same hash function.
3. The new hash code and the decrypted hash code are compared. If the hash codes match, the recipient has verified that the data has not been altered and also that the sender is who they claim to be.

The hash function is extremely sensitive to changes in data. If the data had been altered in any way, the new hash code it produced would not have been identical to the original hash code. So the matching hash codes prove that the data has not been altered since the signature was created. Note that a digital signature only ensures data against modification, but it does not prevent somebody from viewing the data. So we also have to encrypt the data to prevent the data from unauthorized viewing.

2.1.2 Certificates

There is a problem with using public and private keys to encrypt and sign data. The question is how we can know if the public key we are using belongs to the right person. Certificates can solve this problem. A certificate contains the basic information detailing a person's identity and his/her public key.

2.1.3 Certification Authority

A certification authority (CA) is a trusted entity that issues certificates. The certification authority certifies the authenticity of users and guarantees the information in the certificate is valid. Similar to a passport, a user's certificate is issued and signed by a Certification Authority and acts as proof that the correct public key is associated with the user. Therefore, through third-party trust, anyone trusting the Certification Authority can also trust the user's key.

2.1.4 Public Key Infrastructure

Public key cryptography, certificates, and a certification authority collectively form a security management system. Such a system is called a public key infrastructure (PKI). The PKI enables users of an insecure public network such as the Internet to securely and privately exchange data. It provides integrity of digitally signed data and protection of encrypted data. The PKI is the underlying technology that provides security for the Secure Sockets Layer (SSL) and Hypertext Transfer Protocol Secure Sockets (HTTPS) protocols, which are used extensively to conduct secure E-business over the Internet. It also provides several security services including:

1. Enabling trust through a Certification Authority

   The CA enables trust by certifying the authenticity of users and their key pairs.

2. Certificate retrieval from a certificate repository

   The users of PKI can search for and retrieve the public keys contained in certificates from a certificate repository.

3. Certificate revocation

   Certificate revocation is a process of informing users when the information in a certificate is no longer valid. Certificate revocation mechanisms consist of publishing a certificate revocation list (CRL). A CRL is a list of revoked certificates that is signed and issued by a CA.

4. Key backup and recovery

   The CA can keep a copy of the users' private keys in case they are lost. If a user loses his/her private decryption key, the CA can recover the encrypted data using the copy.

5. Automatic update of key pairs and certificates

   Key pairs and certificates only have a limited lifetime. The PKI can automatically update the key pairs and issue new certificates when key pairs and certificates expire.

6. Non-repudiation

   The PKI can guarantee that only the person whose digital signature appears on a piece of data could have signed that data. A person can not deny that he or she digitally signed the data.

2.2 XML Signature

2.2.1 XML Signature Overview

XML Signature is a specification which the W3C and IETF (Internet Engineering Task Force) jointly proposed for encrypting data and tags within an XML document. This specification was designed specifically for XML.

An XML signature is a digital signature expressed in XML. Like any other digital signature, an XML signature supports authentication, data integrity and non-repudiation to the data that it signs. However, unlike the digital signature standards, an XML signature allows for signing just

part of an XML document. And also it allows for attaching multiple signatures to different portions of the document.

2.2.2 XML Signature Structure

Listing 2-1 shows the basic structure of the <Signature> element.

```
<Signature>
    <SignedInfo>
        <CanonicalizationMethod/>
        <SignatureMethod/>
        <Reference URI?>
            <Transforms/>
            <DigestMethod>
            <DigestValue>
        </Reference>
    </SignedInfo>
    <SignatureValue>
    (<KeyInfo>)
</Signature>
```

Listing 2-1 XML Signature Structure

A brief explanation of the elements used in an XML signature is as follows:

<Signature>	contains the signature, it is the parent element for the signature.
<SignedInfo>	contains the information about the data that are signed.
<CanonicalizationMethod/>	specifies the algorithm used to transform the data into canonical XML.
<SignatureMethod/>	specifies the algorithm used to generate the signature.

<Reference URI>	each resource to be signed has its own <Reference>, identified by the URI attribute
<Transforms/>	lists the processing steps applied to the data before it was digested
<DigestMethod>	specifies the digest algorithm applied to the data.
<DigestValue>	contains the actual digest.
<SignatureValue>	contains the actual signature value.
<KeyInfo>	indicates the key to be used to validate the signature.

An actual XML signature example is shown in Listing 2-2.

```
<Signature Id="MyFirstSignature" xmlns="http://www.w3.org/2000/09/xmldsig#">
    <SignedInfo>
        <CanonicalizationMethod
            Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
        <SignatureMethod
            Algorithm="http://www.w3.org/2000/09/xmldsig#dsa-sha1"/>
        <Reference URI="http://www.w3.org/TR/2000/REC-xhtml1-20000126/">
            <Transforms>
                <Transform
                    Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
                </Transforms>
                <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
                    <DigestValue>j6lwx3rvEPO0vKtMup4NbeVu8nk=</DigestValue>
                </DigestMethod>
        </Reference>
    </SignedInfo>
    <SignatureValue>MC0CFFrVLtRlk=...</SignatureValue>
    <KeyInfo>
        <KeyValue>
            <DSAKeyValue>
                .....
            </DSAKeyValue>
        </KeyValue>
    </KeyInfo>
</Signature>
```

Listing 2-2 XML Signature Example [Bartel et al. 2002]

In this example, the data object that is to be signed is identified by the URI attribute in the Reference element. It is an HTML document here. As specified in the SignatureMethod element, the digital signature is generated using the DSA-SHA1 algorithm. We compute the document's digest value with the digest method SHA1 which is specified in the DigestMethod element. Finally the document is signed by the signature over the digest value.

2.2.3 XML Signature Generation

In this section, we briefly discuss how to create an XML signature. For complete information, please refer to the XML Signature specification (http://www.w3.org/TR/xmldsig-core/).

1. Find the URIs of the resources that are to be signed.
2. Produce a digest for each resource.
3. Specify each digest, along with the algorithms and transforms together in a Reference element.
4. Collect the content in the Reference elements with information about the signature algorithm and canonicalized algorithm to form the SignedInfo section.
5. Canonicalize the SignedInfo data.
6. Calculate the digest of the SignedInfo data and calculate the signature of the SignedInfo digest.
7. Add the signature value to the SignatureValue element.
8. Insert the key information into the KeyInfo element.
9. Place the SignedInfo, SignatureValue , and KeyInfo elements into a Signature element.

2.2.4 XML Signature Validation

The following is a brief description of how to verify an XML signature:

1. Verify the signature of the SignedInfo element. First, re-calculate the digest of the SignedInfo element by using the digest algorithm specified in the SignatureMethod element. Then use the public verification key to verify that the value of the SignatureValue element matches the digest of the SignedInfo element.
2. If step 1 passes, re-calculate the digests of the references contained within the SignedInfo element and compare them to the digest values expressed in each Reference element's corresponding DigestValue element.

2.3 XML Encryption

2.3.1 XML Encryption Overview

XML encryption is a specification developed by W3C. The specification defines a process of encrypting and decrypting digital content. It has two main characteristics. One is that the encrypted content can be represented in XML. The recipient of the encrypted content can use XML DOM (a parser) to extract the cipher text and feed it into an algorithm for decryption to recover the XML content. The other characteristic is that portions of a document can be selectively encrypted. This latter capability is unique to XML encryption.

With XML encryption we can use both symmetric and asymmetric cryptography to secure data. Moreover, XML encryption is interoperable with XML Signatures. If we want to encrypt and sign a document, we have to encrypt the document before we sign it. This is because the digest,

generated for the digital signature, may give clues about the unencrypted content of a document [Vordel].

2.3.2 XML Encryption Structure

The structure of XML encryption is shown in Listing2-3. Please note that "?" denotes zero or one occurrence; "*" denotes zero or more occurrences. We can see the XML Signature element below, distinguished by its "ds" namespace.

```
<EncryptedData Id? Type? MimeType? Encoding?>
    <EncryptionMethod/>?
    <ds:KeyInfo>
        <EncryptedKey>?
        <AgreementMethod>?
        <ds:KeyName>?
        <ds:RetrievalMethod>?
        <ds:*>?
    </ds:KeyInfo>?
    <CipherData>
        <CipherValue>?
        <CipherReference URI?>?
    </CipherData>
    <EncryptionProperties>?
</EncryptedData>
```

Listing 2-3 XML Encryption Structure

The <EncryptedData> element specifies the encrypted text. The <encryptionMethod> element specifies what kind of encryption method was used, the <KeyInfo> element contains the

information about the key that was used to encrypt the data, and the <CipherData> element contains the raw encrypted data. The encrypted data is either the CipherValue element's content or the CipherReference element's URI attribute.

2.3.3 XML Encryption Types

Due to the capability of encrypting only portions of a document using XML Encryption, there are five types of encryption:

- Encrypting an XML Element
- Encrypting XML Element Content (Elements)
- Encrypting XML Element Content (Character Data)
- Encrypting Arbitrary Data and XML Documents
- Super-Encryption: Encrypting EncryptedData

Refer to the W3C XML Encryption Syntax and Processing for complete information about these five encryption types. I now provide an example of encrypting an XML element.

2.3.4 Encrypting an XML element, example

Listing 2-4 shows that Alice has a credit card with a limit of $6,000CAD. We know that Alice's credit card number is sensitive information. We can keep it confidential by encrypting the entire CreditCard element or only the data content of the Number element. Listing 2-5 shows an example where the CreditCard element has been encrypted. Note that an EncryptedData block replaced the CreditCard element and its contents. The CipherData element contains the actual encrypted CreditCard element.

```
<?xml version='1.0'?>
    <PaymentInfo xmlns='http://UM.edu/details'>
        <Name>Alice</Name>
        <CreditCard Limit='6,000' Currency='CAD'>
            <Number>1234 5678 1234 5678</Number>
            <Issuer>ScotiaBank</Issuer>
            <Expiration>12/06</Expiration>
        </CreditCard>
    </PaymentInfo>
```

Listing 2-4 XML Document for Alice's Credit Card

```
<?xml version='1.0'?>
    <PaymentInfo xmlns='http://UM.edu/details'>
        <Name>Alice</Name>
        <EncryptedData Type='http://www.w3.org/2001/04/xmlenc#Element'
                xmlns='http://www.w3.org/2001/04/xmlenc#'>
            <CipherData>
                <CipherValue>A23B45C56...</CipherValue>
            </CipherData>
        </EncryptedData>
    </PaymentInfo>
```

Listing 2-5 Encrypted CreditCard Element

2.3.5 Encryption Steps

There are seven steps involved in encrypting data using XML encryption [Imamura et al. 2002].

1. Choose an encryption algorithm. With XML encryption we can use both symmetric and asymmetric cryptography to secure data. XML encryption supports Triple-DES and AES, both block-cipher algorithms.

2. Obtain and (optionally) represent the encryption key. The recipient requires a decryption key to decrypt the ciphertext. If the recipient already knows the key, there is no need to include the key with the encrypted data. If the recipient does not know the key, the sender should include it. It can be referred to by name, or URI below the ds:KeyInfo element. If the key itself is to be encrypted, construct an EncryptedKey element by recursively applying this encryption process. The result may then be a child element of ds:KeyInfo, or it may exist elsewhere and may be identified in the preceding step.
3. Serialize the data into UTF-8 encoding. If the data is XML plaintext, we must convert XML into UTF-8 before encryption. UTF stands for Unicode Transformation Format. The serialization makes the plaintext into octets (a sequence of bytes).
4. Perform the encryption. Once we have the octets, we can encrypt the octets using the chosen algorithm and key.
5. Specify the data type. The data type is described with the Type, MimeType and Encoding attribute. For example, if the data is an XML document, we could use MimeType="text/xml" to describe the data type.
6. Build the EncryptedType structure. EncryptedType is the abstract type for both EncryptedData and EncryptedKey. An EncryptedType structure represents the information about the encryption algorithm, parameters, key, and the type of the encrypted data.
7. Process EncryptedData. If the Type of the encrypted data is element or the contents of an element, we construct the EncryptedData element by removing the identified element or content and inserting the EncryptedData element in its place. The encrypted credit

card example shows this process in Listing 2-5, where the EncryptedData element replaced the CreditCard element and its contents.

2.3.6 Decryption Steps

There are five steps involved in decryption. They are the reverse of the encrypting steps.

1. Determine the algorithm, parameters, and ds:KeyInfo element. The recipient processes the element to find what algorithm, parameters, and ds:KeyInfo element are to be used.
2. Locate the key. The key can be located from the ds:KeyInfo element. If the key is encrypted, it can be decrypted using the recipient's private key. Alternatively, the key can be retrieved from a local key store according to its name in a KeyName element.
3. Decrypt the data which is in the CipherData element. First the recipient retrieves the data and obtains the encrypted octet sequence by base64 decoded. Then the recipient decrypts the octet sequence using the algorithm and key obtained from the previous steps.
4. Process the decrypted data of XML elements or XML element content. First the recipient interprets the plaintext octet sequence as UTF-8 encoded character data. Then the character data replace the EncryptedData element according to the XML document context.
5. Process the decrypted data that is not an XML element or XML element contents. If the decrypted data is not an XML element or XML element content, the recipient should return it to the application together with the Type, MimeType, and Encoding attribute values. The application process or interpret the data according to the Type value.

2.4 XML Key Management Specification (XKMS)

2.4.1 XKMS Overview

XML Key Management Specification 1.0 was a technical note submitted to the W3C in March 2001. After that, an XKMS working group was formed to develop a standard. The XKMS 2.0 specification was finally released in April 2004.

XKMS outlines protocols for the distribution and registration of public keys. XKMS supports XML Encryption and XML Signatures. An XKMS specification is made up of two parts. One is the XML Key Information Service Specification (X-KISS) which relates to the processing and validation of public key. The other is XML Key Registration Service Specification (X-KRSS) which deals with the registration and revocation of public keys. Both of these specifications are discussed in the following subsections.

2.4.2 XML Key Information Service Specification

X-KISS supports two services: locating public keys and validating public keys.

2.4.2.1 Location of keys

A locate service functions like a directory. To locate a key, the client sends a locate request to the XKMS service. The request includes a <ds:KeyInfo> element which contains the information about the desired key, for example, a certificate. The locate service resolves the <ds:KeyInfo> element and parses the certificate to get the public key and its binding information, and then sends it back to the client, but the locate service does not verify that the returned information is trustworthy.

Consider a concrete example of the locate service. A client receives a signed XML document which contains a <ds:KeyInfo> element from Alice. The client wants to get Alice's public key to verify her signature. So the client sends a request to an X-KISS locate service for the public key name and key value. Listing 2-6 and Listing 2-7 shows the client request and server response respectively.

```
<Locate>
    <Query>
        <ds:KeyInfo>
            <ds:KeyValue>
            ......
            </ds:KeyValue>
        </ds:KeyInfo>
    </Query>
    <Respond>
        <string>KeyName</string>
        <string>KeyValue</string>
    </Respond>
</Locate>
```

Listing 2-6 Client Locate Request

```
<LocateResult>
    <Result>Success</Result>
        <Answer>
            <ds:KeyInfo>
                <ds:KeyName>O=Trust.org OU="Crypto"
                        CN="Alice"</ds:KeyName>
                <ds:KeyValue>...</ds:KeyValue>
            </ds:KeyInfo>
        </Answer>
</LocateResult>
```

Listing 2-7 Server Locate Response

2.4.2.2 Validation of keys

A locate service does not guarantee the key it returns is trustworthy. The XKMS validate service provides all the functions of locate, and it also validates the returned key according to the policy of the validate service. Now let us continue with the locate example. After the client gets Alice's key name and key value, he or she wants to know if the key belongs to Alice (i.e. if the binding between Alice and her key is trustworthy and valid). Listing 2-8 and Listing 2-9 shows the validate request and response separately.

```
<Validate>
  <Query>
    <Status>Valid</Status>
    <ds:KeyInfo>
      <ds:KeyName>...</ds:KeyName>
      <ds:KeyValue>...</ds:KeyValue>
    </ds:KeyInfo>
  </Query>
  <Respond>
    <string>KeyName</string>
    <string>KeyValue</string>
  </Respond>
</Validate>
```

Listing 2-8 Validate Request

```
<ValidateResult>
  <Result>Success</Result>
  <Answer>
    <KeyBinding>
      <Status>Valid</Status>
      <KeyID>http://Trust.org/assert/20050120-39</KeyID>
      <ds:KeyInfo>
        <ds:KeyName>...</ds:KeyName>
        <ds:KeyValue>...</ds:KeyValue>
      </ds:KeyInfo>
      <ValidityInterval>
        <NotBefore>2005-03-20T12:00:00</NotBefore>
        <NotAfter>2005-04-20T12:00:00</NotAfter>
      </ValidityInterval>
    </KeyBinding>
  </Answer>
</ValidateResult>
```

Listing 2-9 Validate Response

2.4.3 XML Key Registration Service Specification

X-KRSS is used to generate and manage public key credentials. X-KRSS provides four services: register, recover, re-issue, and revoke. These four services together manage the lifecycle of public key credentials.

2.4.3.1 Register

The register service enables the client to register a public key pair with an XKMS service. The public key pair can be generated either by the client itself or through the register service. The client side key generation is more secure than the service key generation because generating the key at the client side avoids the disclosure of the private key to the register service.

- Client Key Generation: A Client requests registration by specifying the information it wants to bind to a public key through a key binding. The service accepts the registration, and returns a successful generation.
- Service Key Generation: A Client requests registration without the public key information. The service response contains both the public key information and the encrypted private key.

2.4.3.2 Re-issue

The reissue service allows previously registered key bindings to be issued again. This is due to the need for periodic updating of the key binding. While the key binding used by XKMS has no lifespan, the credentials issued by the underlying PKI may have a lifespan that needs to be renewed from time to time. The client sends a re-issue request which contains the key binding to be re-issued to the re-issue service. The service returns either the re-issued key binding or a rejection.

2.4.3.3 Revoke

If a private key has been disclosed, the client must inform the XKMS service by requesting a revoke service. The revoke service revokes a previously registered key binding. After that, the key binding is destroyed and no longer considered trustworthy. To send a revoke request, the client should specify that the status of the key binding is 'Invalid'. If the revoke operation is successful, the status of the key binding is changed from 'Valid' to 'Invalid'. If there is no record of the key binding in the registration service, the revoke service returns a 'NotFound' result.

2.4.3.4 Recover

If an important data file is encrypted and a client has lost his/her private key required to decrypt the file, it is necessary for the client to request key recovery. A key recovery system stores a copy of the private key during registration so that it can be recovered. Note that, the recover service can only recover private keys if they are generated at the server rather than at the client during registration.

To send a recovery request, the client should specify the key pair to be recovered using a key binding element and an authentication element. If the recovery operation is successful, the service returns the recovered private key in an encrypted form, and the status of the key binding is changed from 'Valid' to 'Invalid' using the revoke operation. This is done because the registration service policy defines that whenever the key is recovered, it should be revoked for security reasons.

2.5 WS-Security

2.5.1 WS-Security Overview

The WS-Security specification was first released by IBM, Microsoft, and VeriSign in April 2002. It was submitted to a standards body called the Organization for the Advancement of Structured Information Standards (OASIS) in June 2002. Then OASIS formed a web services security group to develop WS-Security as an OASIS standard. In April 2004, OASIS Web Services Security 1.0 (WS-Security 2004) was finally released as a standard.

WS-Security specifies a mechanism for signing and encrypting parts of a SOAP message. It proposes a standard set of SOAP extensions that can be used when building secure Web services to implement message content integrity and confidentiality [WS-Security]. WS-Security is flexible and is designed to support multiple security models, mechanisms, and technologies. It provides three major mechanisms: message integrity, message confidentiality, and ability to pass around security tokens as part of a message. However, it does not support authentication mechanisms, non-repudiation, etc.

2.5.2 IBM/Microsoft Web Services Security Road Map

WS-Security is just the first specification as part of the IBM and Microsoft Web services security road map. The road map includes six other would-be specifications. These specifications are divided into two layers: Policy and Federation. The road map is shown in Figure 2-1.

Policy

- WS-Policy, allows enterprises to specify the security requirements of their Web Services. Such as, algorithms for encryption and digital signatures.
- WS-Trust, defines how trust relationships are established. There are two trust relationships: direct and brokered.
- WS-Privacy, defines a way for enterprises that deploy Web Services to state and communicate privacy policies.

Federation

- WS-Secure Conversation establishes a mutually authenticated security context to allow a requestor and a Web Service to exchange SOAP messages.
- WS-Federation indicates how trust relationships in different security specifications are managed and brokered.
- WS-Authorization describes how access policies for a Web Service are specified and managed.

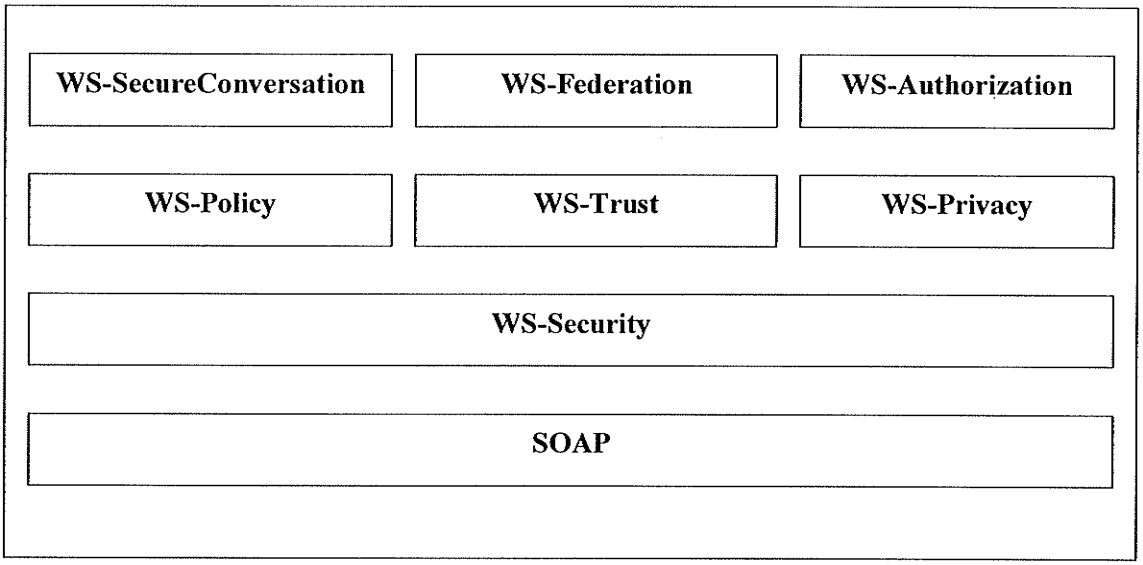

Figure 2-1 IBM and Microsoft Web Services security specifications

Looking at the road map, we can see that the specifications are being produced from the bottom up, with SOAP at base of the diagram. Each specification depends on those of its predecessors.

2.5.3 WS-Security Elements and Attributes

Before discussing elements and attributes, consider the definitions of the security terminology used in WS-Security specifications. The following definitions are referenced by Web Services Security: SOAP Message Security 1.0.

Claim: A *claim* is a declaration made by an entity (e.g. name, identity, key, group, privilege, capability, etc).

Claim Confirmation: A *claim confirmation* is the process of verifying that a claim applies to an entity.

Security Token: A *security token* represents a collection of (one or more) claims.

The WS-Security specification defines a set of XML elements and attributes which can include security tokens in the SOAP messages. It also specifies how to use XML Signature and XML Encryption to protect the confidentiality and integrity of these tokens. There are two kinds of security tokens: username token and binary security token.

The example of Listing 2-10 illustrates the use of a binary security token with an XML Signature. WS-Security defines a <wsse:BinarySecurityToken> element that can be included in the <wsse:Security> block to describe a binary security token.

- The ValueType attribute contains a qualified name that defines the value type of the encoded binary data. In this example, the ValueType is wsse:X509v3 for an X.509 v3 digital certificate.
- The encodingType attribute is used to specify the encoding format of the binary data. Here the data is base64-encoded binary data.

- The Id attribute is a string label, "X509Token" for example security token.

Consider the KeyInfo section that contains a <wsse:SecurityTokenReference> element. The URI attribute of <wsse:Reference> points to the X.509 certificate. The X.509 certificate carries the Id "X509Token" to link it to the XML Signature.

The <ds:SignedInfo> section specifies what is being signed and the type of canonicalization being used. Consider <ds:Reference>, the subelement of <ds:SignedInfo>. The attribute URI= "#MsgBody" points to the "#MsgBody" Id attribute of the StockSymbol element in the SOAP body. This indicates that the <S:Body> element is signed.

```
<?xml version="1.0" encoding="utf-8"?>
<S: Envelope xmlns: S="http://www.w3.org/2001/12/soap-envelope"
             xmlns: ds="http://www.w3.org/2000/09/xmldig#"
             xmlns: wsse="http://schema.xmlsoap.org/ws/2002/07/secext"
             xmlns: xenc="http://www.w3.org/2001/04/xmlenc#">
    <S: Header>
        <wsse: Security>
            <wsse: BinarySecurityToken
                        ValueType="wsse:X509v3"
                        EncodingType="wsse:Base64Binary"
                        Id="X509Token">
                    FHUIORvCCA9CgAwIB...
            </wsse: BinarySecurityToken>
            <ds:Signature>
                <ds:SignedInfo>
                    <ds:CanonicalizationMethod Algorithm=
                            "http://www.w3.org/2001/10/xml-exc-c14n#"/>
                    <ds:SignatureMethod Algorithm=
                            "http://www.w3.org/2000/09/xmldsig#hmac-sha1"/>
                    <ds:Reference URI="#MsgBody">
                        <ds:DigestMethod Algorithm=
                                "http://www.w3.org/2000/09/xmldsig#sha1"/>
                        <ds:DigestValue>LyLsF0Pi4wPU...</ds:DigestValue>
                    </ds:Reference>
                </ds:SignedInfo>
```

```
                <ds:SignatureValue>DJbchm5gK...</ds:SignatureValue>
                <ds:KeyInfo>
                    <wsse:SecurityTokenReference>
                        <wsse:Reference URI="# X509Token"/>
                    </wsse:SecurityTokenReference>
                </ds:KeyInfo>
            </ds:Signature>
        </wsse: Security>
    </S: Header>
    <S: Body>
        <tru:StockSymbol Id="MsgBody"xmlns:tru="http://quotes.com/payloads">
            QQQ
        </tru:StockSymbol>
    </S:Body>
</S: Envelope>
```

Listing 2-10 WS-Security Example

Summary

This chapter provided a brief background review of Web services security. The PKI (Public Key Infrastructure) was introduced first and then some specifications concerning XML security, such as XML Signatures, XML Encryption, and XKMS (XML Key Management Specification) were discussed in more detail. Finally WS-Security for securing SOAP messages was discussed.

Chapter 3

Credit Card Web Service Architecture

3.1 The Client/Server Model and 3-tier Architecture

The term "Client/Server" describes the relationship between two computer programs in which one program, the client, makes a service request to another program, the server, which fulfills the request. In the usual client/server model, one server, sometimes run as a daemon, is activated and awaits client requests. Typically, multiple client programs share the services of a common server program. The Internet offers a typical client/server example where web browsers act as clients which request services (the sending of web pages or files) from web servers located elsewhere on the Internet. The client/server model provides a convenient way to interconnect programs that are distributed across different locations. Most business applications written today use the client/server model and the 3-tier architecture is the most popular way to implement this model.

A 3-tier architecture describes how an application program is organized. Three standard software architecture tiers are: 1) the user interface, 2) business logic, and 3) databases and programming related to managing it. In a typical situation, the application user's workstation contains the first tier in the form of a graphical user interface (GUI) and application-specific data entry forms or interactive windows. The second tier, business logic, is located on a server or other shared computer that acts as the server for client requests from workstations. In turn, it

determines what data is needed (and where it is located) and acts as a client in relation to the third tier that includes the database and a program to manage read-and-write accesses to it. We can generalize to an "n-tier" architecture, which is the same as a 3-tier architecture from a logical point of view, but each tier may have multiple component(s) or server(s) that spread out physically in different network nodes in order to improve performance and scalability.

3.2 The 3-tier Architecture for a Credit Card Web Service

The Credit Card Web Service is built on the 3-tier architecture model. I divide the Credit Card Web Service application into 3 distinct layers: the presentation layer, the business layer and the data layer as shown in Figure 3-1.

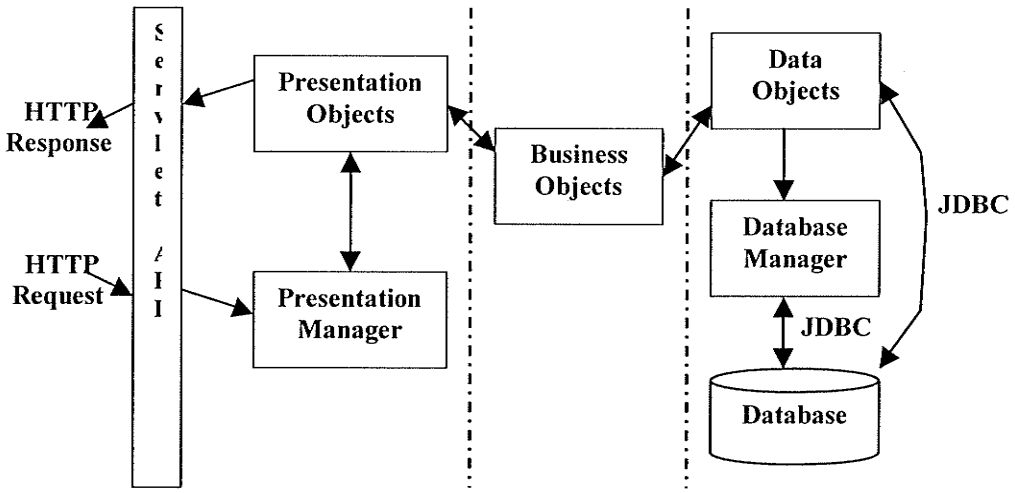

Figure 3-1 3-tier Design in Credit Card Web Service

The Presentation Layer: this layer contains the presentation objects responsible for generating the HTML pages in response to user requests and thus allows the user to navigate the services

of the application. As the name implies, a presentation object presents information to end-users. Dynamic content is normally generated by feeding parameters into the business layer. A presentation object knows where to get the content that is to be presented, but how this content is generated or attained is not the responsibility of presentation objects. In our application, Input.jsp, Method.jsp and Result.jsp are used as the presentation objects to allow the user to input a credit card number and perform the get_limit action and display the result. The presentation manager handles the loading and execution of the presentation objects. Note that there is only one presentation manager in Credit Card Web Service application.

The Business Layer: this layer contains the business objects (BOs), each of which is responsible for a specific business process. In general, a business object is called by other business objects or by presentation objects. A business object normally gets the needed data from the data objects within the data layer and then produces the desired output after executing a series of actions. The CreditCardService.java was defined as a business object which is responsible for validating credit cards.

The Data Layer: this layer contains the data objects (DOs) and the methods used to handle the different data components. A data object is an abstract representation of a logical unit of the application data. For example, a card DO might contain the information for a credit card. It could then contain attributes such as card number, card type, expiration date, etc. Although in many cases data objects correspond to data stored in a database, data objects can also be stored in memory or in a file. Nevertheless, these details are hidden from the other layers.

In the data layer, a database manager controls an application's pool of database connections. It works with a logical database, which is the abstraction that hides the differences between different database types. In the case of the application developed here, the database manager is responsible for establishing the actual connection between the logical database and the actual database through the use of JDBC (Java DataBase Connectivity).

Dividing the application into these three layers greatly improves the modularity of the application and reduces the complexity of future maintenance tasks.

3.3 Use Case Analysis

The Unified Modeling Language (UML) is the industry-standard visual modeling language for specifying, visualizing, constructing, and documenting the artifacts of software systems, as well as for business modeling and other non-software systems [Booch et al. 1999][D'Souza and Wills 1998]. UML uses a few diagram types to model the structure of systems, these include sequence diagrams, class diagrams and use case diagrams.

Use Case analysis is useful in identifying system requirements and domain objects. A Use Case is a description of a set of sequences of actions that a system performs to yield an observable result or value to an actor [Booch et al. 1999]. An actor represents a coherent set of roles including human users, hardware devices, or other external systems that interact with the system.

We used the UML modeling technique to describe the secure credit card system development. In our credit card checking scenario, there are four actors: the client, browser, controller and database server. The client can perform check_credit_card_number; check_expiration_date; and check_limit actions. The browser receives these actions and displays the results which it gets from the controller. The database server actor can perform retrieving number, date and limit action and send the retrieved data values to the controller. The controller actor can display these data action. Figure 3-2 illustrates the credit card checking scenario use case diagram.

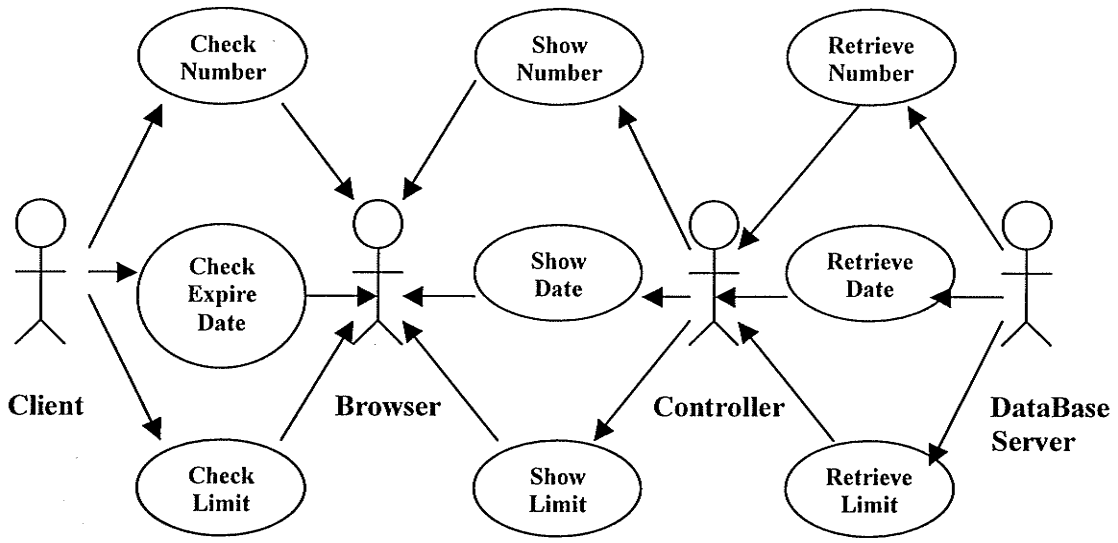

Figure 3-2 Credit Card Checking Scenario Use Case Diagram

3.4 Entrust PKI

Entrust Technologies is one of the vendors that offers PKI product one of which is called Entrust/PKI.

3.4.1 Entrust PKI Architecture [Toolkit]

Entrust/PKI is a collection of applications that work together to make up the PKI. The core components of Entrust/PKI are Authority, Authority Master Control, RA, Authority database, and the Directory. Figure 3-3 shows an overview of the relationships among these core components of Entrust/PKI.

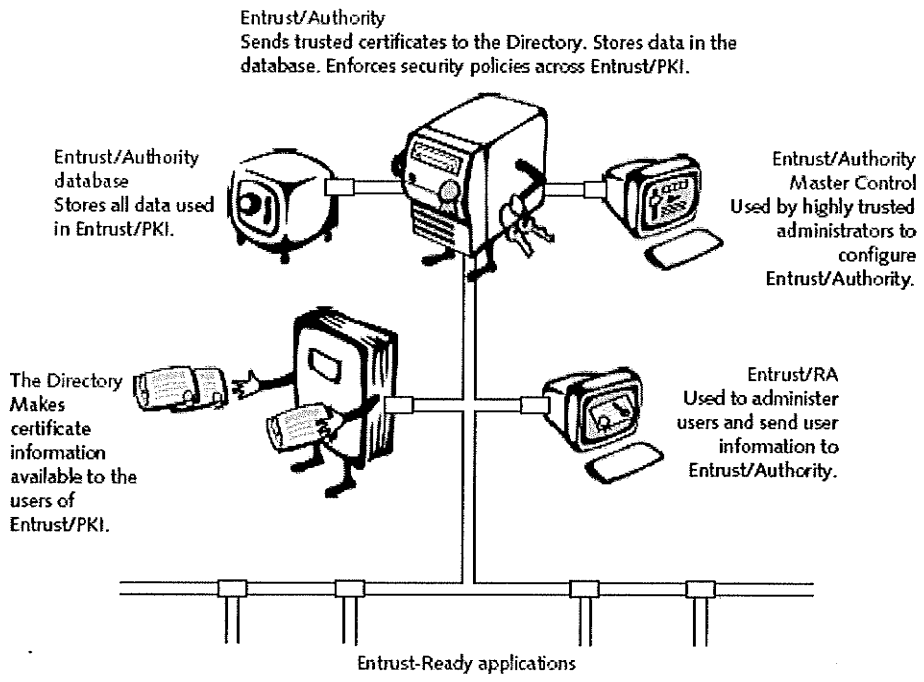

Figure 3-3 Entrust/PKI core components and their relationships [Toolkit]

Authority

Authority is the center of the Entrust/PKI. It plays the role of a Certification Authority (CA). The CA issues, manages and revokes certificates. It manages a secure database to perform key backup, recovery and update. In addition, it supports cross certification with other vendors' CA products; this feature allows a company which is using Entrust software to verify certificates issued by other vendors to ensure they are valid and secure. For example, my credit card web service application will use Entrust/PKI as the underlying security infrastructure. When a client of this web service needs to check a credit card, he or she may send a request containing a certificate which is issued by another vendor, say VeriSign. Our application which uses the Entrust/PKI can then validate the certificate.

Authority Master Control

Authority Master Control is a local interface with direct access to the Authority. It provides trusted access to Authority similar to that of the most highly trusted administrators. Running in either command-line or GUI form, the Authority Master Control is used to perform the tasks that include: starting and stopping the Authority service; recovering profiles for Security officers; and managing the Authority database.

RA

The RA (Registration Authority) is the administrative component of the Entrust/PKI. RA uses a graphical interface and communicates securely with the Authority entity. The RA is used for administrative tasks that include: adding users; managing users and their certificates; managing

security policies; cross-certifying with other Certification Authorities; and setting up hierarchies of Certification Authorities.

The Authority database

The Authority database is under the control of the Authority and acts as a secure storage area for all information related to the Entrust/PKI. In this database Authority, it stores: key pairs which are signed by CA; user status information; the encryption key pair history for each user; the verification public key history for each user; and revocation information, etc. All information stored in the Authority database is protected against tampering, with all sensitive information being encrypted.

The Directory

The majority of user requests for information involve retrieving other users' certificates. To make this information publicly available, the Entrust/PKI uses a public repository known as a Directory. Information that is made public through the Directory includes: user certificates, lists of revoked certificates; client policy information. The Directory is the most frequently accessed component during information requests across the Entrust/PKI.

3.4.2 Entrust/PKI User Roles

Entrust/PKI provides a division of responsibilities to ensure security. Supporting this division is a variety of distinct user roles, capable of carrying out a full range of tasks within the Entrust/PKI. The default administrator roles in the Entrust/PKI are "Master User", "Security

Officer", "Administrator", "Directory Administrator", and "Auditor". The default non-administrator role is "End User". The functions of these roles are as follows:

Master User: performs system-level operations involving the Authority entity. This is the only user who can use the Authority Master Control.

Security Officer: uses the RA to administer sensitive Entrust/PKI operations.

Administrator: primarily administers End Users.

Directory Administrator: modifies information listed in Entrust/PKI's Directory.

Auditor: can view audit logs, reports and user properties but not modify then.

End User: is a non-administrative Entrust user. End Users cannot log into an Entrust/RA. They must have a certificate for use within the Entrust/PKI.

3.5 PKCS #7: Cryptographic Message Syntax Standard

3.5.1 Overview

The Public-Key Cryptography Standards (PKCS) are offered by RSA Laboratories to promote the development of secure applications and other standards based on public-key cryptography. First published in 1991, PKCS has become widely implemented and referenced, and a significant amount of experience is now available to assist the development of related formal standards, as well as for the improvement of PKCS itself [Kaliski and Kingdon 1997].

PKCS #7 is a Cryptographic Message Syntax standard which provides a particular example of PKCS at work. This standard describes a general syntax for data that may have cryptography applied to it, such as digital signatures and digital envelopes. The syntax supports recursion so

that we can use multiple levels of digital signatures and envelopes. It also allows arbitrary attributes, such as signing time, to be authenticated along with the content of a message, and provides for other attributes such as countersignatures to be associated with a signature [PKCS7].

This standard supports six different content types: 1) Data, 2) Signed Data, 3) Enveloped Data, 4) Signed-and-Enveloped Data, 5) Digested Data and 6) Encrypted Data. If both the sender and receiver use the PKCS #7 standard to implement digital signatures or digital envelopes, it is not necessary for them to exchange algorithm information. They would be able to exchange data distinctly.

3.5.2 Encryption and Decryption Sequence

PKCS #7 will be used in my Credit Card Web Services application to encrypt and digitally sign sensitive information, such as credit card number. Figure 3-4 shows the sequence diagram for the Encryption/Decryption process. The implementation of PKCS #7 will be explained in more detail in Chapter 5.

In the sequence diagram, A represents a client of the Credit Card Web Service, and B represents the Web Services application. Clients want to make requests for checking credit cards. PKCS #7 makes it possible to transit the sensitive information such as credit card number over an insecure network. The following elaboration describes the encryption and decryption sequence between a client and our application:

1. *A* signs on to the CA (Certificate Authority) with a valid profile (e.g. p1.epf). *A* requests the public key for *B* (represented by p2.epf).
2. The CA checks that p1 and p2 are valid. The CA responds with the public key for *B*. The requester can then verify that *B*'s public key has been signed by the CA.
3. *A* formats the message as specified, encrypts the results with *B* 's public key and signs it with *A*'s private key.
4. These results are embedded within an XML message to be sent to *B*.
5. Upon receipt of the XML, *B* signs on to the CA with valid profile (p2.epf) and requests the public key for *A*.
6. The CA checks that p1.epf and p2.epf are valid. The CA responds with the public key for *A*. The requester can then verify that *A*'s public key has been signed by the CA.
7. Using the public key, *A*'s signature is verified and using *B*'s private key (held in the local certificate) the message is decrypted.

Note that profiles, such as p1.epf and p2.epf, are the certificates of the holders in Entrust PKI context. This Entrust profile contains the user's public and private credentials, such as the public encryption and verification key, private decryption key, and private signing key.

Encrypting sensitive data prevents unauthorized viewing and applying a digital signature assures the data has not been modified in any way, as well as verifying the identity of the sender. Data that is both encrypted and signed provides for the maximum security.

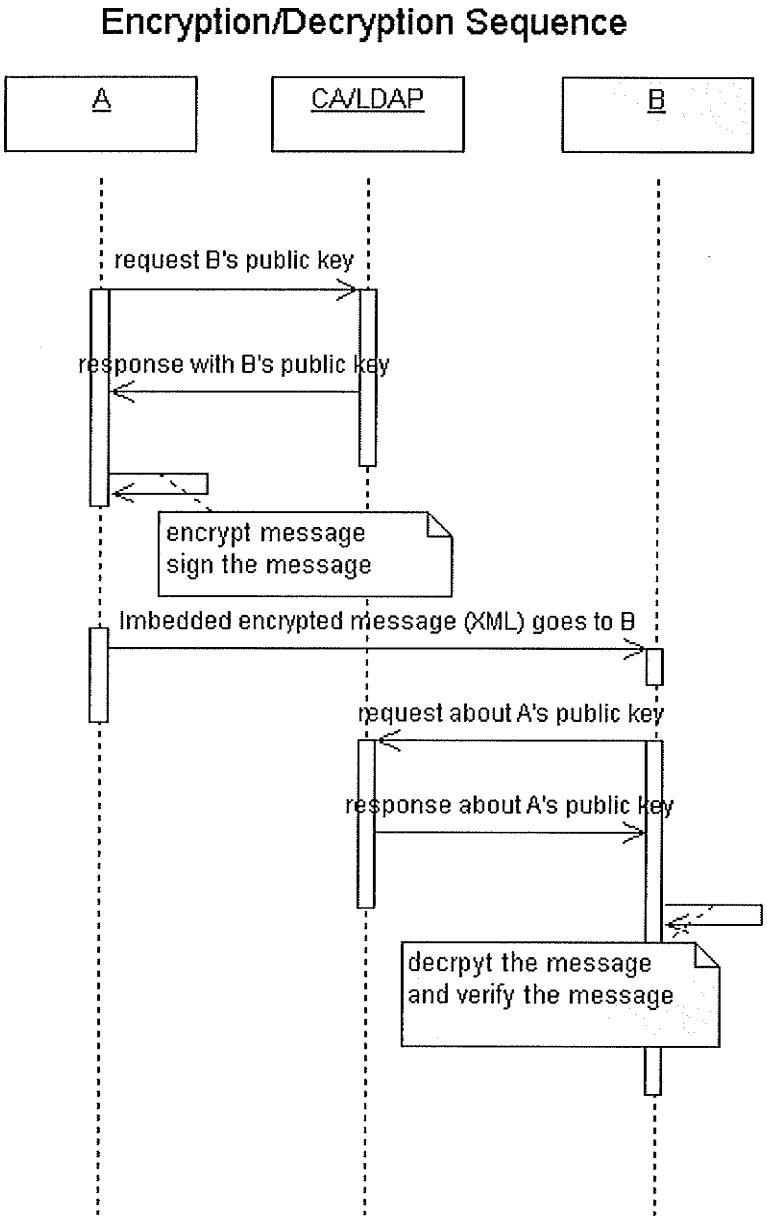

Figure 3-4 Encryption/Decryption Sequence Diagram

Summary

This chapter gave an overview of the client/server model and 3-tier architecture used in my Credit Card Web Service. An in-depth description of the prototype development, use case analysis was also discussed. The Entrust PKI, chosen as the underlying security architecture, was also discussed. Finally, the PKCS #7 cryptography standard was reviewed and its processing highlighted using an encryption and decryption sequence diagram.

Chapter 4

Building the Credit Card Web Service using SOAP, WSDL and UDDI

Having discussed the architecture of the Credit Card Web Service, this chapter will focus on how I built the Credit Card Web Service using SOAP, WSDL and UDDI, existing Web Services standards.

4.1 Overview

An application can be both a service provider as well as a service requestor when using service-oriented architectures. As such, an application can make available some of its functions to others while the same application can also use services published by other service providers.

The Credit Card Web Service provides credit card validation and limit check business functions based on a supplied credit card number. The Credit Card Web Service is also a Web Services consumer. It consumes other Web Services such as the "update card" service and "cancel card" service provided by banks or other financial organizations.

Taking the credit card validation service as an example. The client sends a validation request by providing an encrypted and signed card number. This service decrypts the data, verifies the signature and returns the validation information for the card. The information includes the limit

of the card if it is valid. Based on this exchange, the following sections examine the standards used to implement our Credit Card Web Service. These standards are SOAP over HTTP, WSDL as the Web Services description format, and UDDI as the Publish/Discovery standard.

4.2 SOAP

SOAP is a lightweight XML-based protocol used for the exchange of information in a decentralized, distributed environment [Wolter 2001]. It enables language and platform agnostic communication. It consists of three parts: "an envelope that defines a framework for describing what is in a message and how to process it; a set of encoding rules for expressing instances of application-defined data types; and a convention for representing remote procedure calls and responses."

As SOAP is used for information exchange, SOAP messages are transmitted in a request/response pattern. A SOAP container is an implementation of SOAP which accepts incoming SOAP requests and dispatches them to published components. The SOAP container automatically translates the requests between SOAP and the components' native language. SOAP containers are compatible with many programming languages.

4.2.1 SOAP Message Structure

Every SOAP message is wrapped in a unique envelope. Each envelope consists of a Body section and an optional Header section. Listing 4-1 shows a SOAP request for the getLimit service. And Listing 4-2 shows a SOAP getLimit response.

```
<soap:Body>
    <m:getLimitRequest xmlns: m="http://tempuri.org/um.edu.CreditCardService">
        <cardNo xsi:type='xsd:string' >ATKEKDL...</cardNo>
    </m:getLimitRequest>
</soap:Body>
```

Listing 4-1 SOAP Request

```
<soap:Body>
    <m:getLimitResponse xmlns: m="http://tempuri.org/um.edu.CreditCardService">
        <Limit>3000.00</Limit>
    </m:getLimitResponse>
</soap:Body>
```

Listing 4-2 SOAP Response

The main part of the message is the SOAP Body. In Listing 4-1, a getLimitRequest SOAP request is sent to the Credit Card Web Service. The request takes a string parameter, an encrypted credit card number. In Listing 4-2, the SOAP getLimitResponse response returns a float – the limit amount of the card.

The SOAP Header is an optional part that can be used for different purposes. As was discussed regarding WS-Security in Chapter 2, wsse:Security element can be included in the SOAP header to protect the SOAP message. We can also use the SOAP header for transactions and routing, etc. If the Header is present, it must be the first child of the Envelope. We can use the SOAP-ENV: mustUnderstand attribute to indicate whether or not the receiver can ignore the processing of the header.

4.2.2 SOAP Message Encoding

SOAP is a packaging protocol. Its mandate is to provide a standard data encoding scheme. SOAP encoding makes use of types defined in XML schema and creates the necessary mappings for language-specific type definition to ensure interoperability. We show a mapping between Java and XML schema in Table 4-1.

Java Types	XML Schema Types
Boolean	Boolean
Byte	byte
Char	unsignedShort
Short	short
Int	int
Long	long
Float	float
Double	double
String	string
Date	dateTime
BigDecimal	Decimal

Table 4-1 Mappings between Java Types and XML Schema Types

The example in the previous section used primitive types. The "cardNo" in Listing 4-1 is encoded in SOAP as:

<cardNo xsi:type='xsd:string' >ATKEKDL...</cardNo>

where "xsd:string" indicates a mapping from Java type String to XML Schema type string.

More complex data types can also be used in a SOAP encoding. SOAP makes use of xsd:complexType to encode an object according to its class hierarchy and its member elements. SOAP also provides built-in support for arrays.

The SOAP encoding rules take full advantage of XML Schema: offering a consistent mapping between SOAP and different programming languages which serve as the foundation for interoperability in Web Services architecture.

4.2.3 Summary

SOAP effectively hides the details of the implementation of the end points. The Web Services implementation could be a legacy COBOL application, a new EJB system, an interface to a commercial product written in C++, or anything else that can interpret a request message and return an appropriate response. Similarly, the service provider does not need to know anything about the implementation of its service requesters.

Using SOAP for RPC (Remote Procedure Call) extends the power of object-oriented programming to web-based remote objects. It can also be used for exchanging documents containing any kind of XML data. As a whole, SOAP enables code reuse and fosters transparent integration to systems of any type.

4.3 WSDL

Since a Web Service can be written in any language and reside on any platform, there is a need for a systematic and universally understood means of representing the service. WSDL meets this need. A WSDL document provides the necessary details for a service requestor to contact and consume a service. A WSDL document for the Credit Card Web Service is shown in Listing 4-3.

From Listing 4-3, we see that the document is verbose. Luckily, WSDL is usually generated either statically by development tools, or dynamically by the hosted Web Service. Once we know basic keywords defined in WSDL, it is not hard to understand the specification and associate the definitions with the real world service. In this example, an abstract service, "CreditCardService", is defined, with one operation – "getLimit". This operation takes one parameter "encryptedCardNo" (as schema type "string") in the input message which is defined as "getLimitRequest", and returns another parameter "result" (as schema type "float") in the output message which is defined as "getLimitResponse".

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="CreditCardService"
        targetNamespace="http://edu.um.wsdl/CreditCardService/"
        xmlns="http://schemas.xmlsoap.org/wsdl/"
        xmlns:tns="http://edu.um.wsdl/CreditCardService/"
        xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <message name="getLimitRequest">
        <part name="encryptedCardNo" type="xsd:string"/>
    </message>
    <message name="getLimitResponse">
        <part name="result" type="xsd:float"/>
    </message>
    <portType name="CreditCardService">
        <operation name="getLimit" parameterOrder="encryptedCardNo">
            <input message="tns:getLimitRequest" name="getLimitRequest"/>
            <output message="tns:getLimitResponse" name="getLimitResponse"/>
        </operation>
    </portType>
    <binding name="CreditCardServiceBinding" type="interface:CreditCardService">
        <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
        <operation name="getLimit">
            <soap:operation soapAction="" style="rpc"/>
            <input name="getLimitRequest">
                <soap:body
                        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
                        namespace="http://tempuri.org/um.edu.CreditCardService"
                        parts="encryptedCardNo" use="encoded"/>
            </input>
            <output name="getLimitResponse">
                <soap:body
                        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
                        namespace="http://tempuri.org/um.edu.CreditCardService"
                        use="encoded"/>
            </output>
        </operation>
    </binding>
    <service name="CreditCardService">
        <port name="CreditCardServicePort" binding="tns: CreditCardServiceBinding">
            <soap:address location="http://tempuri.org/um.edu.CreditCardService"/>
        </port>
    </service>
</definitions>
```

Listing 4-3 WSDL document of Credit Card Web Service

A complete WSDL document consists of a set of definitions starting with a root definition element followed by six individual element definitions--types, message, portType, binding, port, and service--that describe the services. Explanations of the six elements now follow:

Types: provides the definitions of data types which are contained in the messages exchanged in the service. Data types can be simple, complex, derived, or array types.

Message: represents an abstract definition of the messages that the Web service exchanges. A WSDL document has a message element for each message that is transmitted, and the message element contains the data types associated with the message. In our example, we define the "encryptedCardNo" as a string data type and "result" as a float type in the message element.

PortType: specifies a set of abstract operations. Each operation refers to an input message and output messages. A WSDL document has one or more portType definitions for each Web service it defines. In Listing 4-3, we only have one portType which defines the getLimit operation.

Binding: specifies a concrete protocol and data format specifications for the operations and messages defined by a particular portType. In the above example, we use the binding element to bind CreditCardService to the HTTP protocol and SOAP format.

Port: provides a single network address for a binding, thus assigning an individual endpoint for the service. We set http://tempuri.org/um.edu.CreditCardService as an endpoint for the CreditCardService in our example.

Service: aggregates a set of related ports and defines a logical name for the Web service through its name attribute. In the service element of our example, we defined our service name CreditCardService using the name attribute.

WSDL is extensible to allow description of endpoints and their messages regardless of what message formats or network protocols are used to communicate [Christensen et al. 2001]. In the Credit Card Web Service, WSDL is used to describe the credit card validation services packaged by SOAP. This abstract service definition can be reused later in other implementations that follow the same service description. For example, one can write a C# program to provide the same "getLimit" service through an IIS server that takes in a card number as input and returns the limit to the application requesting the service. In this case, we just need to add one more "service" tag to this WSDL description to indicate the physical location and port for the new service binding.

4.4 UDDI

UDDI (Universal Description, Discovery and Integration) is a platform-independent framework for describing services, discovering businesses, and integrating business services using the Internet [WSDL 2005]. In a service-oriented architecture it plays the role of a Service Registry. There is no pre-requisite for a business to register or use UDDI. Any business can register a UDDI specification.

4.4.1 UDDI Business Registry

A UDDI Business Registry is an implementation of the UDDI specification. Each instance of a public UDDI Business Registry is also called an Operator Site and is a core element in the Web Services infrastructure. The UDDI Business Registry is run by different "node operators" who

are the companies that run an instance of the public UDDI Business Registry. The operators all support a common set of APIs that will ensure the integrity and availability of the information provided. According to the specification, all operators should replicate the registrations across all nodes on a regular basis. A business only needs register its information with one operator, and the rest of them will have the same entry once they have synchronized. Currently, there are several public UDDI operators, including IBM, Microsoft, and SAP among others. Aside from these public UDDI operators, one can host a private UDDI business registry within an enterprise domain for internal services without sharing information with the general public.

4.4.2 Using UDDI to Register and Find Services

UDDI is used to publish and discover Web Services. All the public UDDI operators provide Web GUIs that allow people to register their businesses and services online. In this section we will talk about how to register and find the Credit Card Web Service through the IBM UDDI Business Registry.

The basic steps involved in registering the Credit Card Web Service are:

1. Obtain a user account from the IBM UDDI Business Registry. UDDI only allows authorized individuals to publish or change information within the UDDI business registry.
2. Register the business information (e.g. name, contacts, description, and locators) and get a unique business ID that thereafter represents the business. The business entity, named "UMFinancial" is shown in Figure 4-1.

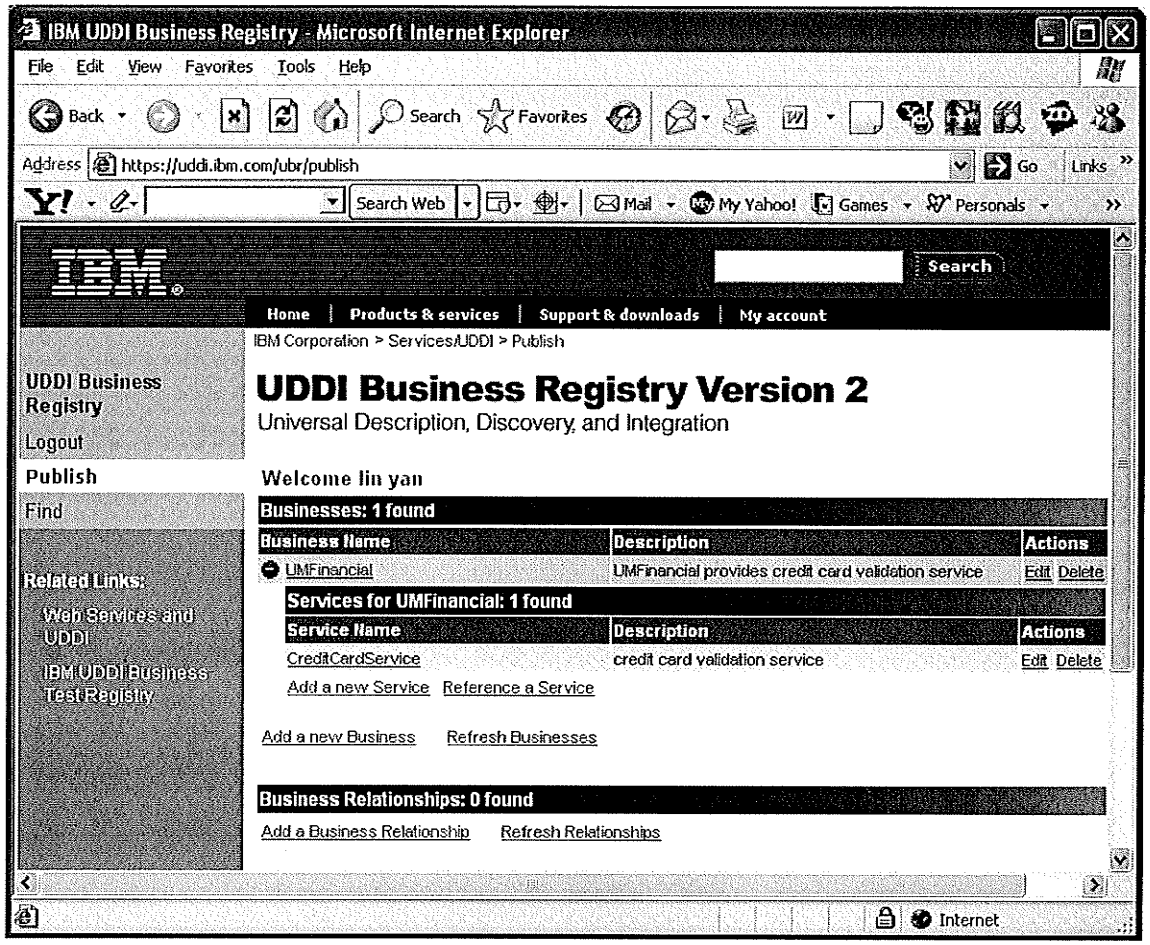

Figure 4-1 A Business Entity in UDDI

A business entity also belongs to one or more categories. The categories used in UDDI include NAICS (North American Industrial Classification System), UNSPSC (Universal Standard Products and Services Classification), and GCS (Geographic Classification System) etc. Thus users can search based on many classification systems. Figure 4-2 shows the details of the registered business.

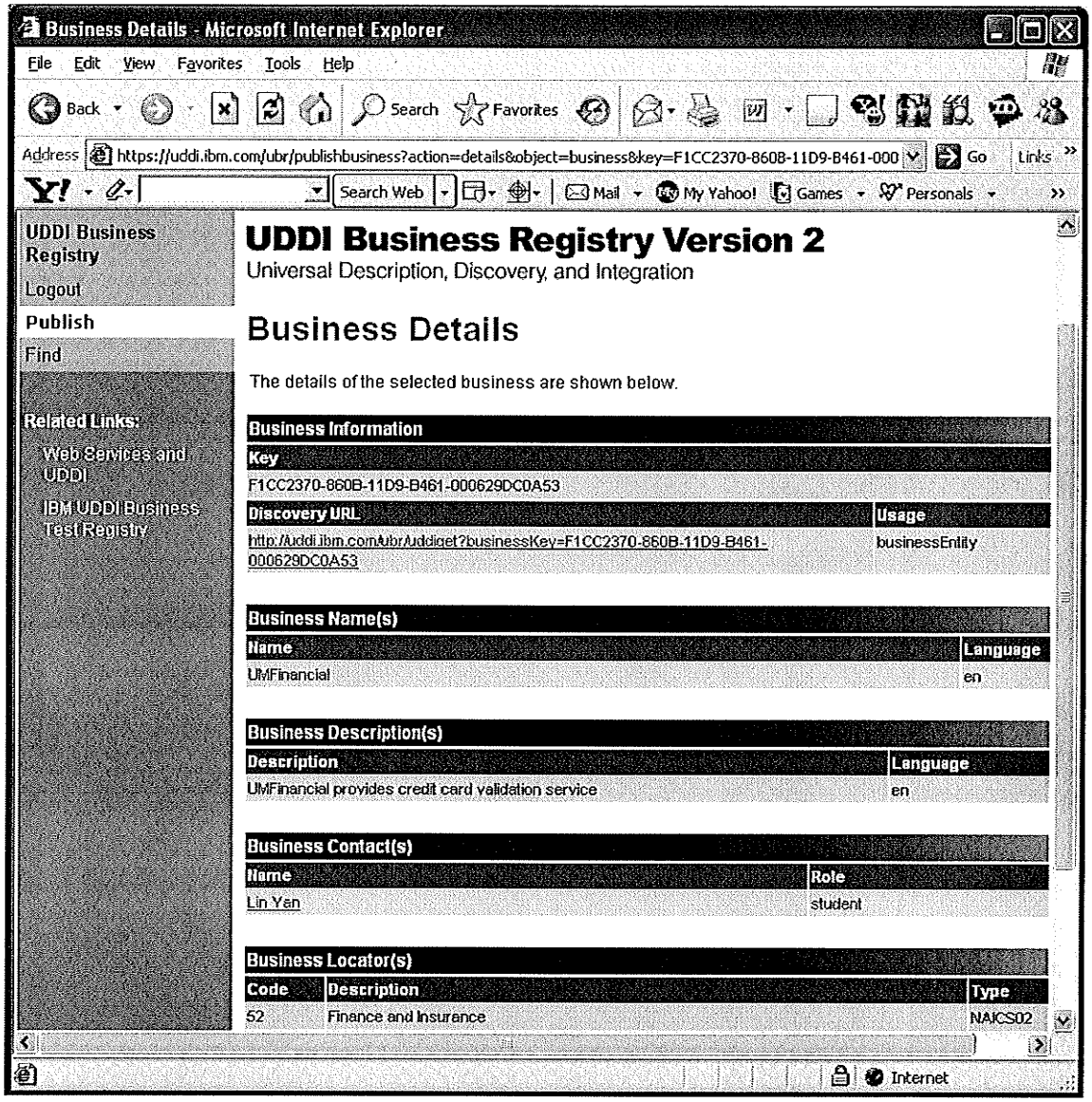

Figure 4-2 Details of a Business

3. Register the Credit Card Web Service. First we define the service to get a unique service ID and then we specify the access point to make it available. Figure 4-3 shows the details of registering the credit card service. Each service must have a unique key, a service name, description, and the owner of the service etc.

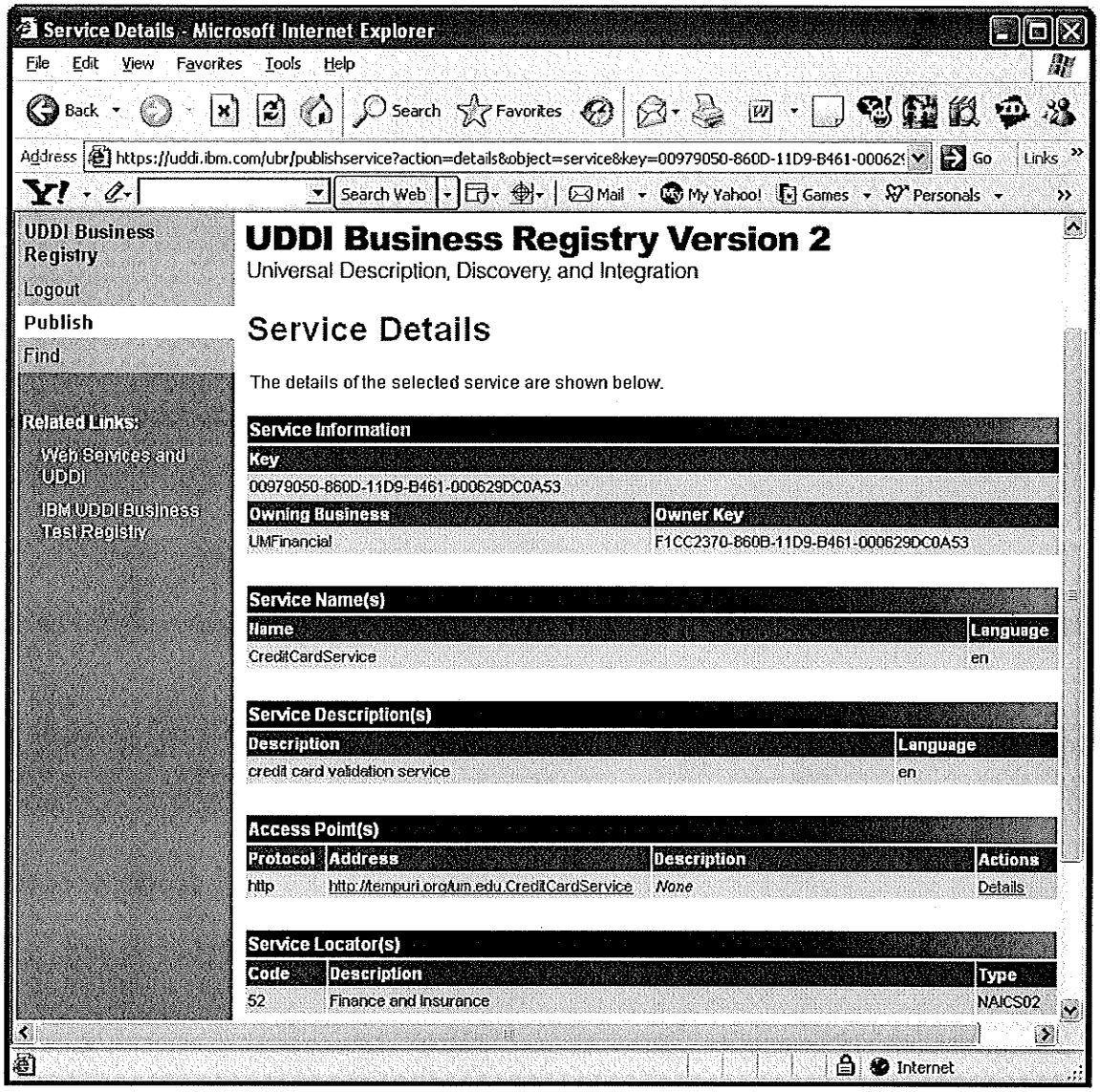

Figure 4-3 Details of a Service

In addition to the interface for registering a business and its related Web Services, the Web interface supported by the operator sites also provides a means for finding a registered business, service, or technical model. The IBM UDDI Find interface is shown in Figure 4-7. Advanced search options are also available where we can search with more specific constraints.

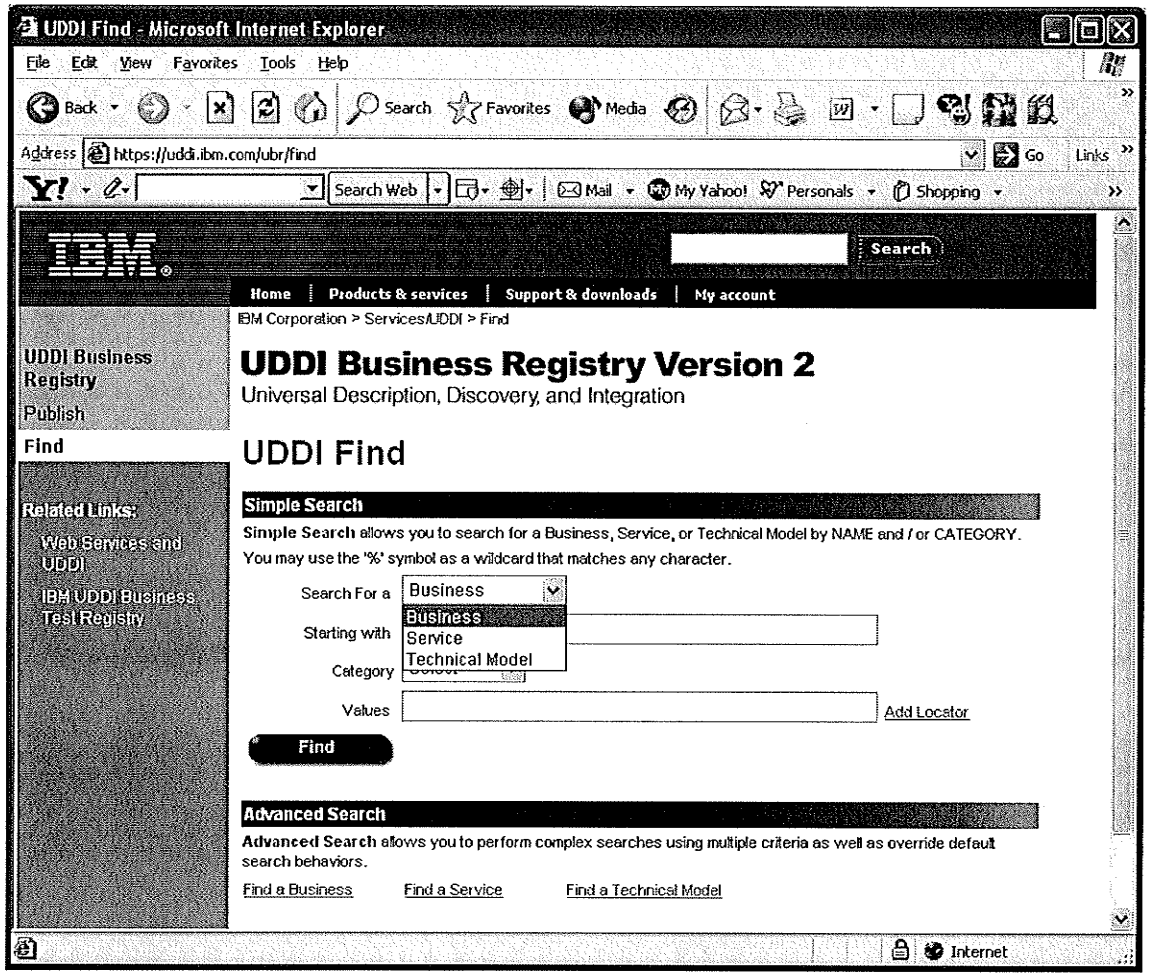

Figure 4-4 UDDI Find Interface

Summary

This chapter discussed how I built Credit Card Web Service using SOAP, WSDL and UDDI. These Web Services standards were explained along with our application.

Chapter 5

Credit Card Web Service Implementation

5.1 Implementation Language

There are many languages available for building Web services applications: PHP, Perl, ASP, Java, etc. We chose Java as the implementation language due to its strong object-oriented support and platform portability.

Java is not designed solely for web application development. It incorporates many standard programming language principles, such as object-orientation, strong typing, and exception handling. It is a language that can be used for both web client and server side programming. Java Applets can also run on browsers, and Java Servlets can run on the server. Java's cross-platform compatibility and convenient APIs for networking and multi-threading have made it popular in the business world.

Java was selected as the implementation language for the Credit Card Web Service for the following reasons:

- Portability: Java is designed to be platform independent. It has the virtue of "compile once, run everywhere". The application should be deployable on any platform providing a Java Virtual Machine (JVM) without a major rebuild.

- Extensibility: The application should have the flexibility to extend its functionality to meet a wide variety of enterprise level requirements. The object-oriented nature of Java makes it easy to extend. In addition, Java also offers a rich set of APIs to implement various tasks.
- Cost Effectiveness: This is a critical factor in the real world. Java is an open source project, which is free to use.
- Performance: Fast response time is important in web applications. Although the performance of a Java program is always an arguable issue, a Java Servlet does have performance advantage over a traditional CGI program since a Java Servlet launches a new thread instead of a new process for each request.

5.2 Implementation Tools

5.2.1 Entrust Authority (TM) Security Toolkit for Java

5.2.1.1 Overview [Toolkit]

The Entrust Authority Security Toolkit for Java, abbreviated the Toolkit here, is one of a number of Entrust Authority Toolkits. The Toolkit uses the Java programming language, and its cross-platform capabilities, to give the ability to add trusted security to our application. It helps one develop and build applications that protect the privacy, integrity, and authenticity of information communicated among the workstations and servers of a network whose security is managed using a Public Key Infrastructure (PKI). It gives our application access to the underlying security structure of a PKI and its automated key and life cycle management capabilities.

The Toolkit offers both high-level and low-level APIs that enable a Java application to perform security related tasks. The Toolkit also adheres to the Java Cryptography Architecture (JCA) (http://java.sun.com/security/) and supplements it with numerous cryptographic algorithms.

Applications which are built with the Toolkit can perform several high-level tasks, such as the creating and processing of PKCS #7 messages. Another task is the management of users' credentials (the user's private and public cryptographic keys, and information that describes a user's relationship to the CA) which involve the creation and recovery of credentials, key rollover, and certificate validation. The credentials themselves will be detailed discussed in section 5.2.1.3.

5.2.1.2 Security Toolkit for the Java Architecture

Java Security provides a collection of Application Programming Interfaces (APIs). Such APIs include the built-in security packages of the Java 2 Software Development Kit (J2SDK), the Java Cryptography Extension (JCE), and the Java Secure Sockets Extension (JSSE). From Figure 5-1, we see the security packages of the J2SDK that underlie the Toolkit and provide the base for cryptographic services. The JCE is the cryptographic framework that underlies the Security Toolkit for Java. The Java Cryptography Architecture (JCA), which governs the design of the security packages of the J2SDK, is extended by the JCE to include APIs for digital signatures, hash functions, encryption, key exchange, and Message Authentication Codes (MAC).

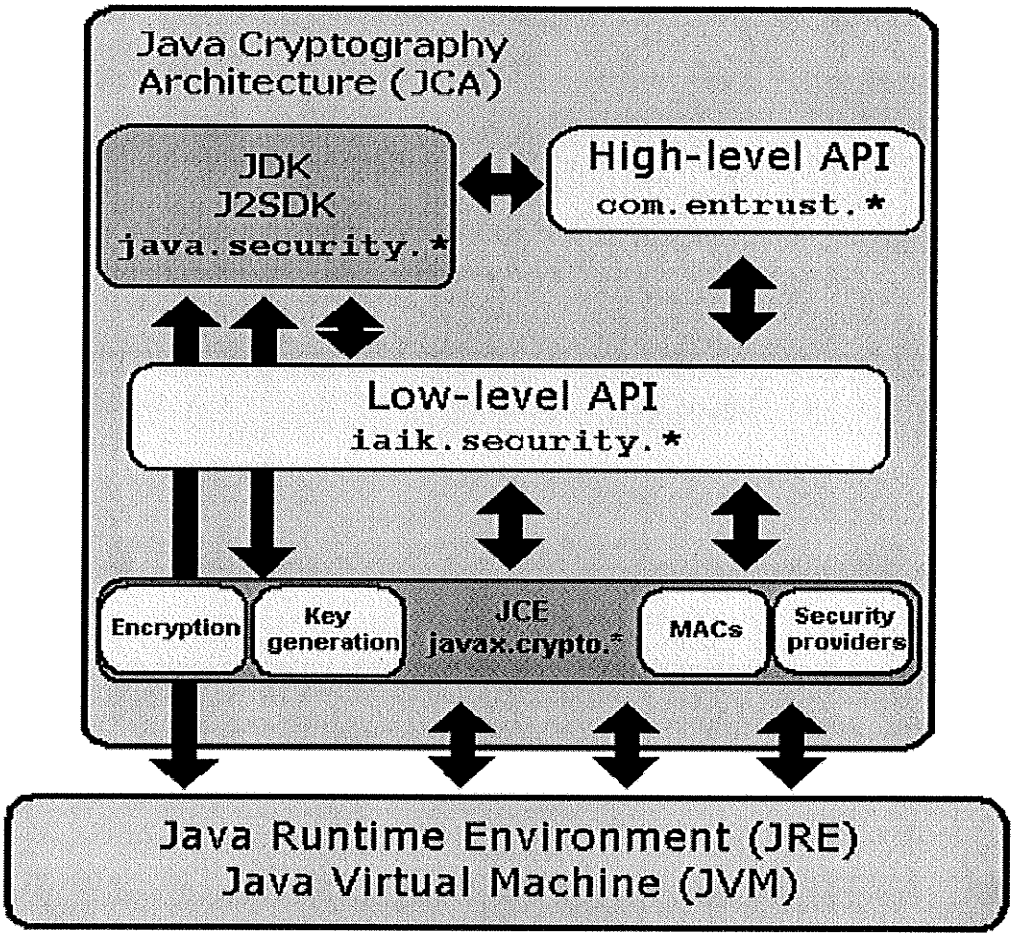

Figure 5-1 Entrust/Toolkit for Java Basic Architecture [Toolkit]

There is a low-level API that resides on top of JCE. The low-level API of the Toolkit comprises the IAIK packages, the *java.security* packages, and the *javax.crypto* packages. Classes in the low-level API support the Public Key Cryptography standards and give us direct access to the JCE and its algorithms, and to the IAIK cryptographic service provider.

The Toolkit's high-level API, found in the *com.entrust.toolkit* package and its subpackages, provides classes that implement frequently used cryptographic tasks. These classes, because of their high level of abstraction, are generally more useful than their low-level IAIK.

The *User* class is the primary class in the high-level API. It represents an entity or an end user in a PKI domain. By instantiating a *User*, we gain access to the user's public verification certificate and public encryption certificate, the user's private decryption key and private signing key.

5.2.1.3 Credentials

A *user* in the context of a PKI, defines an entity that has been identified and approved by a Certification Authority (CA). The term *credentials* is used to describe a set of data that contains a user's critical cryptographic information. In an Entrust PKI, an Entrust Profile is used to contain an Entrust user's public and private credentials. This Entrust Profile is usually stored in a file with an .epf file name extension. The contents of an Entrust Profile include: 1) user's distinguished name 2) user's private decryption key 3) user's private signing key 4) public verification and encryption certificates.

The Security Toolkit for Java regards Profiles as the source of keying information for a particular entity, and decouples the key source from key use by defining separate classes for user management (the User class) and credentials management (the Credentials package).

5.2.1.4 Identifying a User

Before users of applications can encrypt or decrypt messages, encrypt and sign data, or communicate securely over a network, they must identify themselves. A PKI, in turn, must be able to recognize users as trusted members of an organization. A user's credentials contain the information needed to authenticate the user to the PKI and to enable a user to perform cryptographic operations.

The process of logging in involves reading and verifying a user's credentials to ensure that the keys they contain are intact and valid. The following code, used in our Credit Card Web Service, illustrates how to authenticate a user.

1. Obtain the user's password as a SecureStringBuffer.

   ```
   String password = "Tst12345";
   SecureStringBuffer spassword =new SecureStringBuffer(password);
   ```

2. Specify the location of the credentials file.

   ```
   String profile = "D:\\lin\\sample\\Entrust\\workplace\\yanlin.epf";
   ```

3. Instantiate a user.

   ```
   User user = new User();
   ```

4. Instantiate a credential reader. In this case, we use a FilenameProfileReader to pass the location of the yanlin.epf Profile as an argument.

   ```
   CredentialReader cr = new FilenameProfileReader(profile);
   ```

5. Call the login method to complete the user's authentication.

   ```
   user.login(cr, spassword);
   ```

The user has a set of credentials, which is used to complete the log in task. In our case, we use an Entrust Profile yanlin.epf stored on the local computer to perform the log in task. If the user's credentials are valid, he or she can login successfully, otherwise, the user is forced to login to our application. To authenticate and login a user, we must also import the following packages to our application.

com.entrust.toolkit

com.entrust.toolkit.credentials

com.entrutst.toolkit.util

com.entrust.toolkit.exceptions

5.2.2 IBM Websphere Studio

The IBM Websphere Studio Application Developer is one of the products of the IBM Websphere Studio family. We chose WebSphere Studio as our IDE (Integrated Development Environment) because it is a state-of-the-art Java IDE. It also provides development tools to enable the creation, development and deployment of Web services, which are self-contained, modular applications that can be described, published, located, and invoked over the Internet. These tools are based on the expected open, cross-platform standards such as UDDI, SOAP, and WSDL [Websphere].

WebSphere Studio facilitates a few processes to support creating and deploying of Web services-enabled applications. It enables us to create Web services from existing artifacts and deploy them in the Websphere Application Server by using server tools to test services running

locally or remotely. It also allows us to publish Web services to the UDDI business registry and discover the services by browsing the registry.

Figure 5-2 shows the logic flow of the Credit Card Web Service in the context of WebSphere Studio. First we create a Credit Card Web Service from a CreditCardServiceBean. Then we generate a Deployment Descriptor to deploy this Web Service on the server. Third, from the Client side, CreditCardServiceProxy is generated to accept the client requests. Fourth, the Simple Object Access Protocol (SOAP) is used between the client and server to encode invocation parameters and results over HTTP. For example, the getLimit() service is one of the services that the CreditCardServiceProxy provides. When the getLimit() service is invoked with a credit card number, the Credit Card Web Service will return the spending limit associated with this card from the Credit Card Database. The credit card number and the limit amount are transmitted in a SOAP encoded message over HTTP.

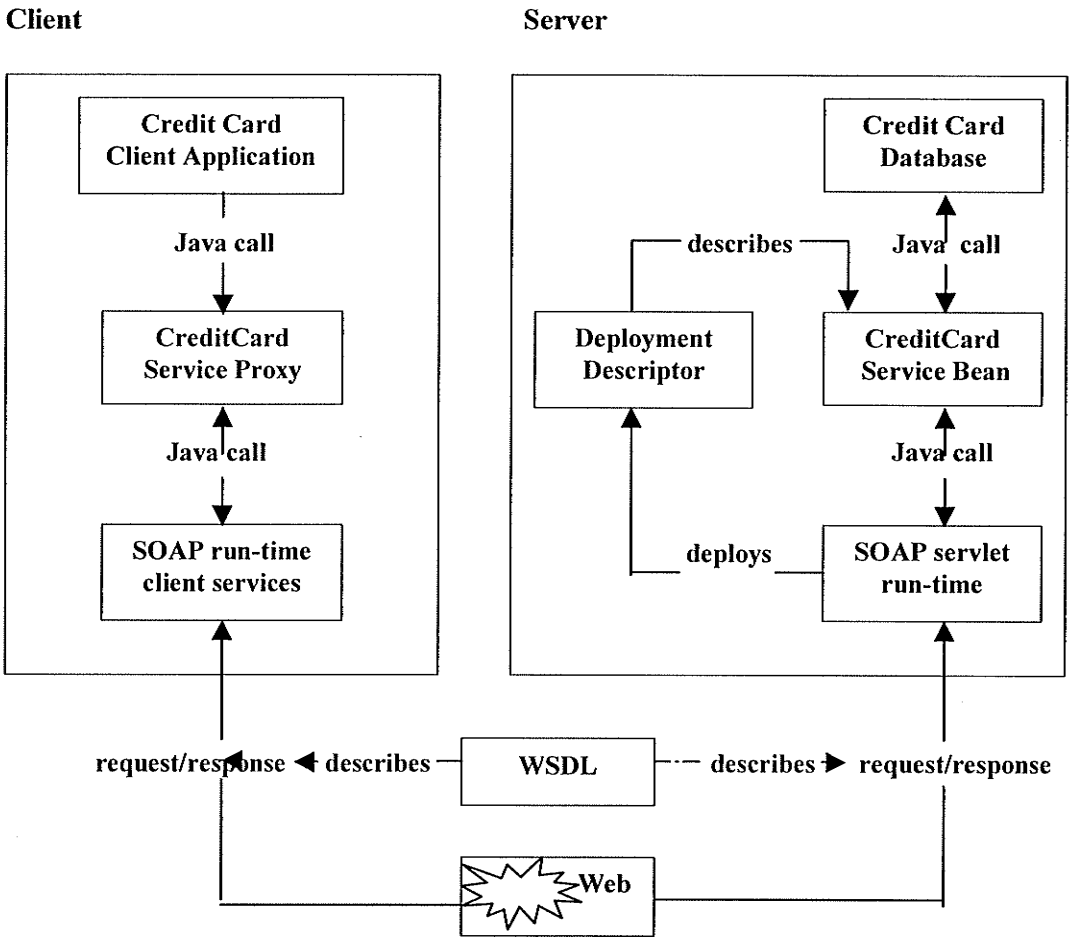

Figure 5-2 Logic flow of Credit Card Web Service in WebSphere Studio

5.3 PKCS #7 Implementation with Entrust Toolkit

In the high-level API, there are two basic PKCS #7 related classes: *PKCS7EncodeStream* and *PKCS7DecodeStream*. These two classes allow us to encode and decode data of arbitrary #7 length, and to verify signatures. *PKCS7EncodeStream* and *PKCS7DecodeStream* extend *FilterOutputStream* and *FilterInputStream* in the java.io package respectively. So data can be processed as it is being read or written.

5.3.1 Encode

The *PKCS7EncodeStream* class signs and encrypts data in accordance with PKCS #7. The following describes the encrypting and signing process in our Credit Card Web Services application.

1. Instantiate, and log in, a user

Please refer to section **5.2.1.4 Identifying a user** for more detail on logging in a user.

2. Create a *PKCS7EncodeStream* object using the *PKCS7EncodeStream* (User, OutputStream, int) constructor:

```
PKCS7EncodeStream encoder = new PKCS7EncodeStream (
                        user,
                        new FileOutputStream(<encryptedCardNo file>),
                        PKCS7EncodeStream.SIGN_AND_ENCRYPT);
```

Where,

- user is a logged in user

- FileOutputStream (<file>) specifies a file that will store the encrypted and signed credit card number from the generated output stream

- PKCS7EncodeStream.SIGN_AND_ENCRYPT is the operation constant indicating that we wish to sign and encrypt data

3. Specify the digest and encryption algorithms we want to use.

For encryption and signing operations the encryption and message digest algorithms are set to CAST and SHA1 by default. We used the default setting in our application. Users can also specify other algorithms by calling the methods encoder.setDigestAlgorithm() and encoder.setEncryptionAlgorithm().

4. Specify the credit card number as the input data and write the encrypted and signed data to the output stream.

```
FileInputStream input_data = new FileInputStream(<location of input data>);
byte[] b = new byte[128];
int i = input_data.read(b);
while (i >= 0)
{
encoder.write(b, 0, i);
i = input_data.read(b);
}
```

5. Close the output stream when the write operation is complete.

```
encoder.close();
```

5.3.2 Decode

The *PKCS7DecodeStream* class decrypts data that has been encrypted according to the PKCS #7 standard, and verifies digital signatures when the end of the input stream is reached. The following describes the decryption process in our Credit Card Web Services application.

1. Instantiate, and log in, a user

Please refer to section **5.2.1.4 Identifying a user** for more detail on logging in a user.

2. Create a PKCS7DecodeStream object using the PKCS7DecodeStream(User, InputStream) constructor.

PKCS7DecodeStream decodeStream = new PKCS7DecodeStream(**user**, **bais**);

Where,

- **user** is a logged in user

- **bais** specifies the input stream (the source of the encoded data), which we get using the following code sequence:

```
byte[] binaryMessage = Util.Base64Decode(encryptedCardNo.getBytes());
ByteArrayInputStream  bais = new ByteArrayInputStream(binaryMessage);
```

3. Read the decrypted and signed data.

```
ByteArrayOutputStream baos = new ByteArrayOutputStream();
int i = decodeStream.read();
while (i >= 0){
        baos.write(i);
        i = decodeStream.read();
        }
```

5 Close the input stream to the encrypted data and the output stream to the newly decoded data.

```
bais.close();
decodeStream.close();
baos.close();
```

5.4 Database Design and Implementation

The IBM DB2 Universal Database was chosen as the DBMS system for the Credit Card Web Service application. The DB2 Universal Database delivers a flexible and cost-effective database

platform for building robust business applications. We can leverage its built-in SQL support, concurrency control, and broad support for open standards, etc.

We established a database with a name CCARD which stores the information about credit cards and card holders. There are two tables defined in the CCARD database. One is the CARDNO table which stores credit card related information, such as credit card number, card type, expiration date, and the card's limit. The other table is the CARDHOLDER table which stores the card holders' related information, such as name, card number, address, telephone, etc. These two tables are linked by the card number attribute.

Various information or data can be stored and accessed from the CCARD database. Database access is through JDBC (Java DataBase Connectivity). JDBC is used to access the DB2 database from the business layer of our application.

Here, at a glance, is the Java code in Credit Card Web Service for connecting to the database:

```
Class.forName("COM.ibm.db2.jdbc.app.DB2Driver");
Connection con = DriverManager.getConnection("jdbc:db2:CCARD","lin", "lin1111");
Statement stmt = con.createStatement();
String sql="select * from CARDNO where NO = '"+ decodedMessage +"'";
ResultSet rs = stmt.executeQuery(sql);
while (rs.next()){
String ccn1 = rs.getString("NO");
limit = rs.getFloat("LIMIT");
}
```

Listing 5-1 Java Code for Connecting to the Database

5.5 Comparison with other Web Services Security Solutions

5.5.1 Benefits and Limitations of Existing technologies

5.5.1.1 SAML

SAML (Security Assertions Markup Language) is an XML-based framework for Web services that provides a standard way to exchange user authentication, authorization and attribute information in an XML document.

SAML includes four main components:

1. Assertions, which are declarations of fact about a subject (human or software entity). SAML assertions includes three types of statements:

   - Authentication statement

     An authentication statement describes that an authority asserts that subject S was identified by authentication method M at time T. This type of statement can be used to enable single sign-on.

   - Attribute statement

     An attribute statement indicates that subject S is associated with some specific details, such as this subject's credit limit or current account paid status.

   - Authorization statement

     An authorization statement identifies whether subject S can be authorized access type A to resource R given evidence E provided by the requesting party.

2. Request/response protocol

SAML defines a request/response protocol to create and exchange assertions. There are two parties involved, one is called the relying party (requesting party), the other is the asserting party (issuing party). The relying party sends SAML assertion request messages, and the asserting party sends back SAML assertion responses which may have one or more assertions with the status of the response.

3. Bindings

   A binding is a way to transport SAML assertion messages to and from authorities. SAML defines rules for mapping SAML assertion messages into real-world messaging or communication protocols.

4. Profiles

   A SAML profile defines constraints and/or extensions of the core protocols and assertions in support of the use of SAML for a particular application. A SAML profile stipulates how particular statements are communicated using appropriate protocol messages over specified bindings [Madsen 2005].

The SAML specification defines how identity and access information is exchanged between communication systems. SAML enables cross-domain trust, which meets the need for one organization to accept identities established by another organization. It applies to three use-case scenarios: single sign-on, distributed transaction, and an authorization service. As Web Services grow and form more and more trust domains, a single trust domain model of identity would not work. While SAML makes security information in the form of assertions about subjects, it

doesn't actually authenticate or authorize users. That's done by authentication, attribute, and authorization authorities. SAML only enables secured data to be transported through several systems for a transaction to be completed.

The Liberty Alliance Project is a major application using SAML, which was developed by a group of vendors and corporate users for providing an enhanced single sign-on standard. Single sign on means that within a federation of connected organizations, if a subject is authenticated by one organization, other organizations within this federation do not require re-authentication. This "portable trust" capability is important for cross-domain B2B Web services to support successfully doing real business with each other. However, due to the standard profiles adapted by the Liberty specification, it is difficult to integrate with other standards-based environments. Further, the Liberty itself is complicated with a large number of profiles and protocols defined by the system.

5.5.1.2 XACML

XACML stands for Extensible Access Control Markup Language, and its objective is to standardize an access control language using XML syntax. It defines action (request) rules for subjects (users) and targets (resources). XACML describes both an access control policy language and a request/response language. The policy language is used to express access control polices (who can do what, where and when). The request/response language expresses queries about whether a particular access should be allowed (requests) and describes answers to those queries (responses) [Lorch et al. 2003].

XACML is consistent with and builds on SAML. SAML's PDP (Policy Decision Point) is implemented with XACML [Rosenberg and Remy 2004]. In the past there has been a number of observations regarding overlaps in functionality between SAML and XACML, with regard to authorization decisions. In the SAML 2.0 Core Specification [Cantor et al. 2004], it states on page 27 regarding the SAML Authorization Decision Statement [XML Research]:

"Note: The <AuthzDecisionStatement> feature has been frozen as of SAML V2.0, with no future enhancements planned. Users who require additional functionality may want to consider the eXtensible Access Control Markup Language [XACML], which offers enhanced authorization decision features."

This is clearly a step toward helping ensure that the two standards do not evolve in an overlapping manner for this functionality. One may interpret this as meaning a brighter future for XACML [XML Research].

As a standard access control language, XACML reduces the cost of developing an application-specific access control language. Plus, system administrators need to understand only one language. If XACML becomes widely used, applications will easily interoperate because they use the same standard access control language. Moreover, XACML is generic so if a policy is written for a specific application, it can be re-used in any other applications. Last but not least, XACML is very extensible. It supports a wide variety of data types, functions, and rules about combining the results of different policies [Brief XACML].

However, XACML is too complicated and probably will be replaced by other development tools. When setting up hierarchical resources with XACML, too much configuration needs to be done. System administrators, who are not familiar with this new access control standard, may find this to be a detriment. Also, the response message is more verbose.

5.5.1.3 Putting Web Services Security Technologies Together

SAML, XACML, XML Encryption, XML Signature, XKMS and WS-Security are all XML-based Web services security standards. We now discuss how these standards could work together.

SAML assertions can be digitally signed using the XML digital signature. The same assertions can be encrypted using XML Encryption to ensure privacy. The public key used for digital signing and encryption can be validated and registered via XKMS. As for XACML, a SAML asserting party could use it to define an access control policy as a basis for handling SAML-based assertion requests [Sang Sing 2003]. Figure 5-3 illustrates how these Web services security standards might work together.

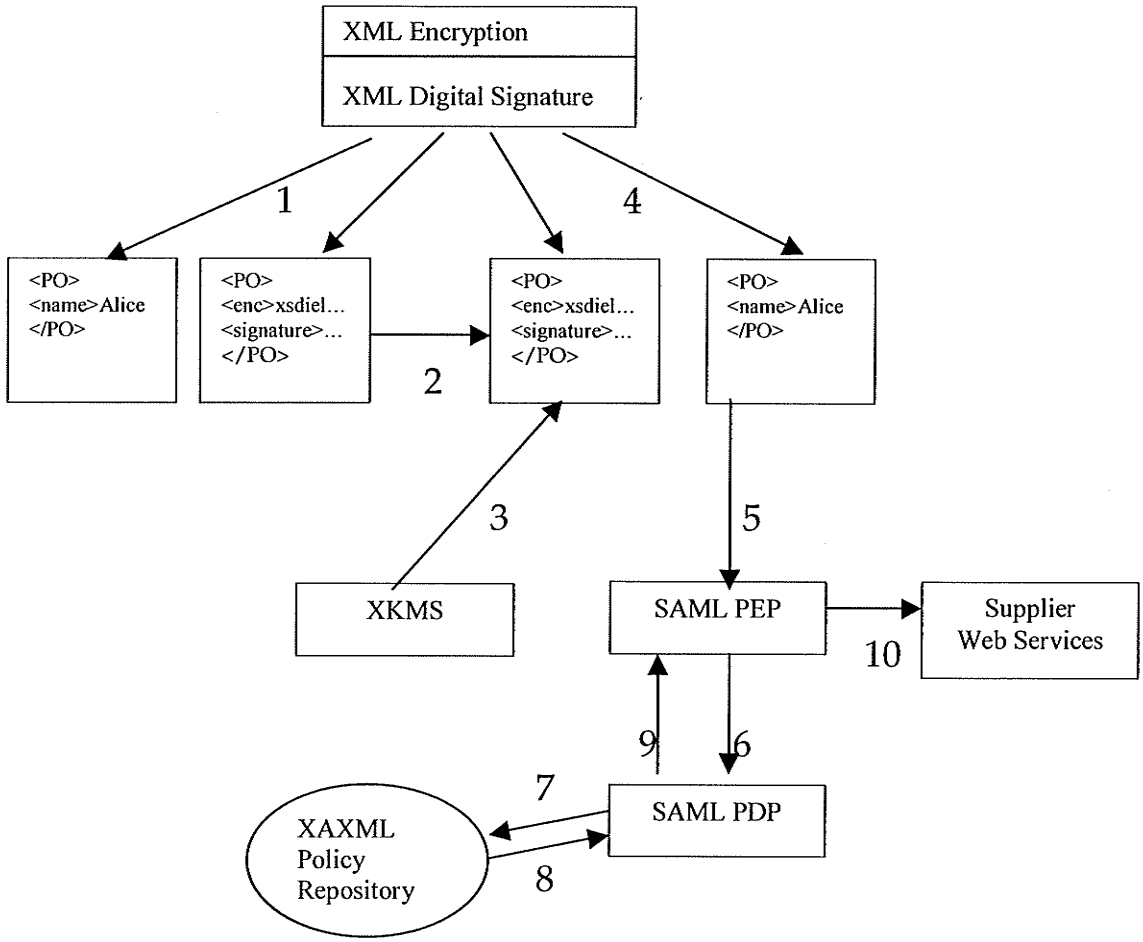

Figure 5-3 Web services security standards work together [Sang Sing 2003][Han Tao 2005]

The following steps describe the sequences where by a client, Alice could send a purchase request to her supplier to use the supplier's web services [Sang Sing 2003][Ardagna 2004] [Han Tao 2005][Anderson and Lockhart 2004][Jeong et al. 2004]:

1. Alice uses XML digital signatures and encryption to digitally sign and encrypt the purchase order XML document.

2. She then sends the document to her supplier, perhaps using SOAP, whose header structure is defined either using the WS-Security or ebXML Message Service standards.
3. The document receiver could then use XKMS to look up and validate Alice's public key.
4. Once the key is determined to be trustworthy, the receiver could then validate and decrypt the purchase order.
5. The receiver might then send an access request to a SAML PEP (Policy Enforcement Point) for the Web service.
6. The SAML PEP would generate the XACMLAuthzDecisionQuery message and send it to the SAML PDP (Policy Decision Point).

7, 8 The SAML PDP would then check a policy repository to obtain one or more XACML policies about Alice. The policy repository would return an XACMLPolicyStatement message which includes the values of subject attribute, resource attribute, action attribute and other parameters.

9. The SAML PDP then must evaluate the policies, and a XACMLAuthzDecisionStatement message is generated as a SAML response
10. Finally, according to the XACMLAuthzDecisionStatement, the SAML PEP would decide if Alice's access request for her supplier's Web service should be permitted or denied.

XML Encryption, XML Signature, XKMS, SAML, XACML and WS-Security are new emerging technologies. These Web services security specifications are not yet true standards. Organizations are still working on them to make them more mature. Also, with these security

specifications, adding the necessary security information into the SOAP header would increase the overhead which may affect the efficiency of transactions between users and web services over the network.

The syntax and the processing model of XML encryption and XML signature are complex. It is difficult to work without a proper implementation support. Also identity collisions may occurr when encrypted contents or XML signatures generated in one context are dropped in another context.

The XKMS 2.0 specification was finally released in April 2004. It is still a specification, and not a full standard. While XKMS has made an attempt to abstract away the complexities of working with a PKI, it is still up to the industry to offer portable and interoperable implementations with easy-to-use APIs to allow for widespread adoption and success [Galbraith et al. 2002].

WS-Security is the foundation for a broader road map for proposed Web services security as outlined by IBM and Microsoft. The proposed road map, discussed in Section 2.5.2, outlines additional web services security specifications the companies plan to develop along with industry partners and standards organizations. Web services security specifications are in the evolutionary stage and have a long way to go. We look forward to seeing more mature WS security specifications in the future.

5.5.2 Benefits and Limitations of the Proposed Solution

Compared with the new technologies for Web Services security, we chose a mature technology, PKI, as our basic underlying security infrastructure and Entrust/PKI as the PKI implementation. PKI is the fundamental component of Web services security architecture. It is most suitable for larger companies with the required financial resources and expertise. Such customer typically wants to provide financial services to the public (e.g. a credit card company). PKI meet these need since it can let the companies to build their own security system. Entrust PKI allows companies to act as their own Certificate Authority (CA), and provides other data security features, such as confidentiality, authentication, non-repudiation, integrity and automatic key management.

Data confidentiality allows only authorized decryption and encryption of data. Data authentication makes sure that the digital signature of the person who signed some data really came from that person. Non-repudiation guarantees a person cannot deny involvement in a transaction that they have digitally signed. Data integrity ensures the protected data is unchanged. A valid digital signature on a piece of data guarantees that the data has not been altered since it was signed. Entrust/PKI software automatically manages the keys of encrypting, signing, decrypting and verifying. Users need not worry about the details of keys. According to these security features Entrust/PKI provided, the mature Entrust/PKI architecture makes our credit card web service application relatively easy to develop and provides sufficient security.

However, PKI also has some drawbacks. One of the drawbacks is the complexity of discovery and validation of the certification paths [Polk and Hastings]. Because isolated CAs can be

combined to form larger PKIs, this makes finding and validating a certificate more complicated. Another drawback is cost. Since PKI allows companies to build their own CAs. The administration cost for issuing, revoking and replacing certificates is significant. Finally, the challenge facing the industry today is how to build and manage circles of trust. As the use of certificates expands, the issue of how individuals should manage multiple certificates to interact with different companies will be a growing concern [Brumley et al. 1999].

5.5.3 Comparison with other Web Services Security Standards

Here, at a glance, is a table showing the specific capabilities are offered by each Web Services Security standard.

Standards	Capabilities
PKI	• Provides integrity of digitally signed data and protection of encrypted data • Enables trust through a Certification Authority • Certificate retrieval and revocation • Key backup, update and recovery • Non-repudiation
XML Signature	• Supports authentication, data integrity and non-repudiation • Allows for signing part of an XML document • Allows for attaching multiple signatures to different portions of the document
XML Encryption	• Encrypted content can be represented in XML • Portions of a document can be selectively encrypted
XKMS	• Outlines protocols for the distribution and registration of public keys • Supports XML Encryption and XML Signature • X-KISS is used to locate and validate public keys • X-KRSS is used to generate and manage public keys
WS-Security	• Allows for signing and encrypting parts of a SOAP message • Supports multiple security models • Supports message content integrity and confidentiality
SAML	• Enables cross-domain trust, which applies to single sign-on, distributed transaction, and an authorization service
XACML	• Describes both an access control policy language and a request/response language • Generic and extensible, supports a wide variety of data types, functions, and rules

Table 5-1 Comparison with other Web Services Security Standards

Summary

This chapter discussed the implementation of our credit card web service. First the implementation language - Java was introduced. Then the use of the Entrust Authority security toolkit for java and Websphere studio were discussed as our application implementation tools. Third, the PKCS #7 standards implementations were explained using example java code. Finally, we compared other Web services security technologies with our PKI solution.

Chapter 6

Conclusions

In this thesis work, we designed and developed a Credit Card Web Service using SOAP, WSDL and UDDI – standards defined in the Web Services community. We also presented a viable approach for securing the Credit Card Web Service through the use of PKI and PKCS #7 standards. The purpose of this approach is to increase the security of transferring XML messages over the Internet. Furthermore, we drew a comparison between this approach and the new emerging specifications for Web Services security, such as XML Encryption, XML Signature, XKMS, WS-Security, SAML and XACML.

Credit Card Web Service application meets the security demands for trust and confidentiality in data transmission, which is achieved by adopting PKI as the underlying security infrastructure. PKI accomplishes these objectives through embedded policy and technology components. These components determine and identify the roles, responsibilities, constraints, range of use, and services available. Also PKCS #7 is used to support digital signatures and document encryption. This viable approach presents an essential way to secure Web Services over a non-secure network.

Because of the limitations of available time and resources, I have not implemented the alternative solution which was discussed in Section 5.5.1.3 in this thesis work. With more

mature new emerging Web Services Security specifications come from OASIS, this issue is worth our further investigation in the future. Now SAML, XACML and WS-Security are domiciled at OASIS, these security approaches should be able to work together nicely and eventually give enterprises some quality choices about how to secure Web services.

Bibliography

[Anderson and Lockhart 2004] Anne Anderson, Hal Lockhart. "SAML 2.0 profile of XACML." OASIS TC Working Draft, Dec 6, 2004.

[Ardagna 2004] Ardagna. "A comparison of modeling strategies in defining XML-based access control languages", Computer Systems Science and Engineering, v19, n 3, May, 2004, pp141-149.

[Bartel et al. 2002] Mark Bartel, John Boyer, Barb Fox, Brian LaMacchia, and Ed Simon. "XML-Signature Syntax and Processing." W3C Recommendation 12 February 2002. http://www.w3.org/TR/xmldsig-core/.

[Bellwood et al. 2002] Tom Bellwood, Luc Clement, David Ehnebuske, Andrew Hately, and Maryann Hondo, et al. "Univeral Description Discovery and Integration Technical Committee Specification Version 3.0." OASIS Open July 2002. http://uddi.org/pubs/uddi-v3.00-published-2020719.htm#_Toc42047184.

[Booch et al. 1999] Grady Booch, James Rumbaugh, and Ivar Jacobson. "The Unified Modeling Language User Guide." Addison Wesley 1999.

[Box et al. 2000] Don Box, David Ehnebuske, Gopal Kakivaya, Andrew Layman, Noah Mendelsohn, Henrik Frystyk Nielsen, Satish Thatte, and Dave Winer. "Simple Object Access Protocol 1.1." W3C Note 08 May 2000. http://www.w3.org/TR/2000/NOTE-SOAP-20000508/.

[Brief] "A Brief History of Internet." Institutionen för informationsvetenskap.

http://www.dis.uu.se/dsv/education/courses/vt05/datavetenskap/misc/A%20Brief%20History%20of%20Internet.pdf.

[Brief XACML] "A Brief Introduction to XACML." http://www.oasis-open.org/committees/download.php/2713/Brief_Introduction_to_XACML

[Brumley et al. 1999] Sanford Brumley, John Jorden and Harry Elmanawy. "Pioneers …or Guinea Pigs?" http://infosecuritymag.techtarget.com/articles/1999/roundtable.shtml

[Cantor et al. 2004] Scott Cantor, John Kemp, Eve Maler. "Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) V2.0." http://xml.coverpages.org/ni2004-08-19-a.html

[Christensen et al. 2001] Erik Christensen, Francisco Curbera, Greg Meredith, Sanjiva Weerawarana. "Web Services Description Language (WSDL) 1.1." W3C Note 15 March 2001. http://www.w3.org/TR/wsdl.

[D'Souza and Wills 1998] Desmond Francis D'Souza, Alan Cameron Wills. "Objects, Components, and Frameworks with UML." Addison-Wesley 1998.

[Entrust] "Entrust PKI and You." Entrust Technologies. 2001. www.entrust.com/resources.

[Galbraith et al. 2002] Ben Galbraith, Whitney Hankison, Andre Hiotis, Murali Janakiraman, Prasad D. V., Ravi Trivedi, David Whitney and Vamsi Motukuru. "Professional Web Services Security." Wrox Press Ltd., 2002.

[Han Tao 2005] Han Tao. "A XACML-based Access Control Model for Web Service."

[Imamura et al. 2002] Takeshi Imamura, Blair Dillaway, Ed Simon. "XML Encryption Syntax and Processing." W3C Recommendation 10 December 2002. http://www.w3.org/TR/xmlenc-core/.

[Jeong et al. 2004] Jongil Jeong, Dongkyoo Shin, Dongil Shin. "An XML-based single sign-on scheme supporting mobile and home network service environments." Consumer Electronics, IEEE Transcations on Volume 50, Issue4, Nov. 2004, pp:1081-1086.

[Joshi et al. 2001] James B.D. Joshi. Walid G. Aref, Arif Ghafoor, and Eugene H. Spafford. "Security Models for Web-Based Applications." Communications of ACM 44.2 February 2001.

[Kaliski and Kingdon 1997] Burton S. Kaliski Jr., Kevin W. Kingdon. "Extensions and Revisions to PKCS #7." An RSA Laboratories Technical Note May 13, 1997.

[Kearney et al. 2004] P Kearney, J Chapman, N Edwards, M Gifford and L He. "An overview of Web Services Security." BT Technology Journal 22.1 January 2004.

[Lorch et al. 2003] Lorch. M., Kafura. D, Shah. S. "An XACML-based policy management and authorization service for global resources." Grid Computing, 2003.

[Madsen 2005] Paul Madsen. "The Building Blocks of Federated Identity" http://www.xml.com/pub/a/2005/01/12/saml2.html

[Monson-Haefel 2004] Richard Monson-Haefel. "J2EE Web Services." Addison-Wesley, 2004.

[O'Neill 2003] Mark O'Neill. "Web Services Security." McGraw-Hill/Osborne, 2003.

[Paxson 1994] V. Paxson. "Growth Trends in Wide-Area TCP Connections." IEEE Network 8.4 July 1994 pp8-17.

[PKCS7] "PKCS #7: Cryptographic Message Syntax Standard." An RSA Laboratories Technical Note Version 1.5 November 1, 1993.

[PKI/Encryption 2000] http://main.uab.edu/show.asp?durki=32241

[Polk and Hastings] WIliam T. Polk and Nelson E. Hastings. "Bridge Certification Authorities: Connecting B2B Public Key Infrastructures."

[Roadmap] "Security in a Web Services World: A Proposed Architecture and Roadmap." IBM and Microsoft April, 2002. http://www-106.ibm.com/developerworks/webservices/library/ws-secmap/.

[Rosenberg and Remy 2004] Jothy Rosenberg, David L.Remy. "Securing Web Services with WS-Security." Sams Publishing, 2004.

[Sang Sing 2003] Sang Sing. "Secure Web services." http://www.javaworld.com/javaworld/jw-03-2003/jw-0321-wssecurity.html

[Singh et al. 2004] Inderjeet Singh, Sean Brydon, Greg Murray, Vijay Ramachandran,

Thierry Violleau and Beth Stearns. "Designing Web Services with J2EE 1.4 Platform." Sun. Microsystems Inc., 2004.

[Toolkit] "Security Toolkit for Java Programmer's Guide." Entrust Inc. August 2003.

[Vacca 2004] John R. Vacca. "Public Key Infrastructure Building Trusted Applications and Web Services." Auerbach Publications, 2004.

[Vordel] "Vordel Tutorial: XML Security." http://www.vordel.com/knowledgebase/tutorial_xml_security/XMLS11.html.

[WebSphere] "IBM WebSphere Studio Application Developer Integration Edition Version 5.0 Getting Started." IBM Corp.2003.

[Wolter 2001] Wolter, Roger. "XML Web Services Basics." Microsoft Corporation. 2001. http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnwebsrv/html/webservbasics.asp.

[WSDL 2005] "WSDL and UDDI." Refsnes Data 2005. http://www.w3schools.com/wsdl/wsdl_uddi.asp.

[WS-Security] "Web Services Security: SOAP Message Security 1.0." WS-Security 2004. OASIS Standard 200401, March 2004. http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0.pdf.

[XML 2004] "Extensible Markup Language (XML) 1.0 (Third Edition)." W3C Recommendation 04 February 2004.

http://www.w3.org/TR/REC-xml/#sec-intro.

[XMLDSig 2005] "XML Digital Signature." Infomosaic Corporation 2005. http://www.infomosaic.net/XMLSignature.htm.

[XML Research] http://lists.xml.org/archives/xml-dev/200409/msg00109.html